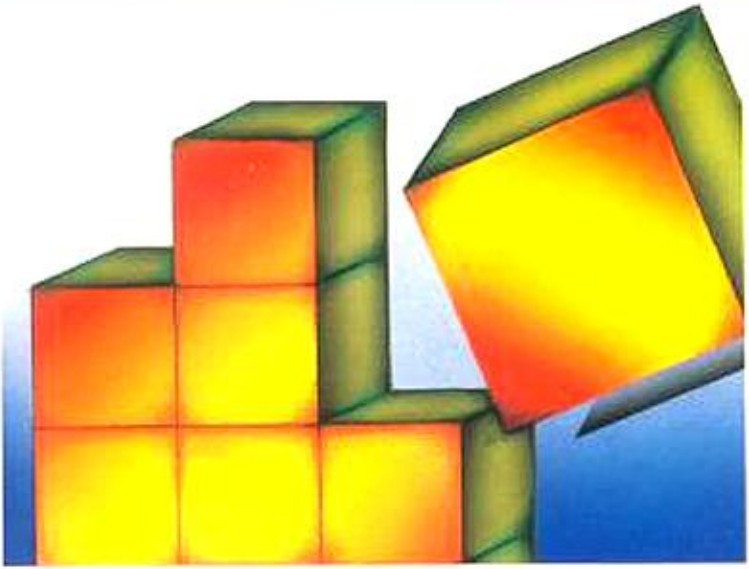
TO 7 MO5 TO7

INITIATION AU FORTH

S-E-F-I



TO TEK international

cedic
nathan

INITIATION AU FORTH

S.E.F.I.

TOTEK International

cedic
nathan

Dans la même collection

Initiation au BASIC TO7 — *Christine et François-Marie Blondel.*

Le BASIC D.O.S. du TO7 et du MO5 — *Christine et François-Marie Blondel.*

Un ordinateur à la maison — *Jean Delcourt.*

Un ordinateur en fête — *Serge Pouts Lajus.*

Un ordinateur et des jeux — *Jean-Pascal Duclos.*

Logo - manuel de référence — *Doris Avram - Tristan Savatier - Michèle Weidenfeld*

Initiation à Logo — *Doris Avram, Michèle Weidenfeld et l'équipe de S.O.L.I.*

Manuel Technique du TO7 et du TO7-70 — *Michel Oury*

Guide du MO5 — *André Deledicq*

Faites vos jeux en assembleur sur TO7 et TO7-70 — *Michel Oury.*

Manuel de l'assembleur du 6 809 et du TO7 et TO7-70 — *Michel Weissgerber*

Ce volume porte la référence
ISBN 2-7124-0557-9

Toute reproduction, même partielle, de cet ouvrage est interdite. Une copie ou reproduction par quelque procédé que ce soit, photographie, photocopie, microfilm, bande magnétique, disque ou autre, constitue une contrefaçon passible des peines prévues par la loi du 11 mars 1957 sur la protection des droits d'auteur.

© CEDIC 1984

CEDIC, 32, boulevard Saint-Germain, 75005 - PARIS

Table des matières

Pages

Introduction 7

Chapitre 1 — Premiers pas en FORTH 8
1.1 Les mots 8
1.2 Le dictionnaire 9
1.3 Définir des nouveaux mots 11
1.4 Quelques mots simples 12
1.5 Supprimer des mots du dictionnaire 13
1.6 Mots en plusieurs exemplaires 14
1.7 Les commodités 15

Chapitre 2 — Une pile de nombres 17
2.1 Écritures et affichage des nombres 18
2.2 La pile des données 18
2.3 Opérations arithmétiques fondamentales 19
2.4 Notation conventionnelle des actions sur la pile 21
2.5 Manipulations sur la pile 21
2.6 Opérations en chaîne 23
2.7 Autres opérations 25
2.8 FORTH et la rapidité 25
2.9 Quelques conseils 26
2.10 Le mot de la fin (du chapitre) 27

Chapitre 3 — Constantes et variables - tables et tableaux 29
3.1 Constantes 30
3.2 Variables 30
3.3 Manipuler des variables 31
3.4 Tableaux 36
3.5 Tables 38

Chapitre 4 — Conditions et répétitions 41
4.1 Expression des conditions 41
4.2 Choisir son action 42
4.3 Avec des si 44
4.4 Combiner des conditions 47
4.5 Répéter une action 48
4.6 Utilisation de BEGIN...UNTIL 52
4.7 Répéter une action un certain nombre de fois 53
4.8 Des boucles imbriquées 56

Chapitre 5 — L'écran en mode texte 59
5.1 Généralités : le mode texte 60
5.2 De toutes les couleurs 63
5.3 Un exemple détaillé 67

Chapitre 6 — L'écran en mode graphique 71
6.1 Des points et des traits 72
6.2 Les couleurs en mode graphique 76
6.3 Des boîtes vides et des boîtes pleines 79
6.4 Le crayon optique 81
6.5 Conclusion sur le mode graphique 82

Chapitre 7 — Récréation : musique 84
7.1 Généralités 85
7.2 La musique en boîte (Play-back) 86
7.3 La musique en mode direct 91

Chapitre 8 — Nombres, caractères et codes 94
8.1 En un octet : les caractères et l'ordinateur 94
8.2 En deux octets : les nombres standard 102
8.3 En quatre octets : les nombres en double longueur 104
8.4 Mots faisant intervenir des nombres standard et double longueur 106
8.5 Entrée et conversion des nombres 108

Chapitre 9 — Chaînes de caractères 110
9.1 Représentation des chaînes de caractères 111
9.2 Variables chaînes de caractères 112
9.3 Opérations sur les chaînes de caractères 114
9.4 Comparaison de chaînes de caractères 119
9.5 Tableaux de chaînes de caractères 120
9.6 Poser des questions en cours d'exécution 121

ANNEXES

Annexe A — L'édition 124
A.1 Principes généraux 124
A.2 Commandes d'éditeur 125
A.3 Editeur et périphériques 130
A.4 L'éditeur vu du FORTH 131

Annexe B — Les périphériques 134
B.1 L'imprimante 135
B.2 Le lecteur-enregistreur de programmes (LEP)

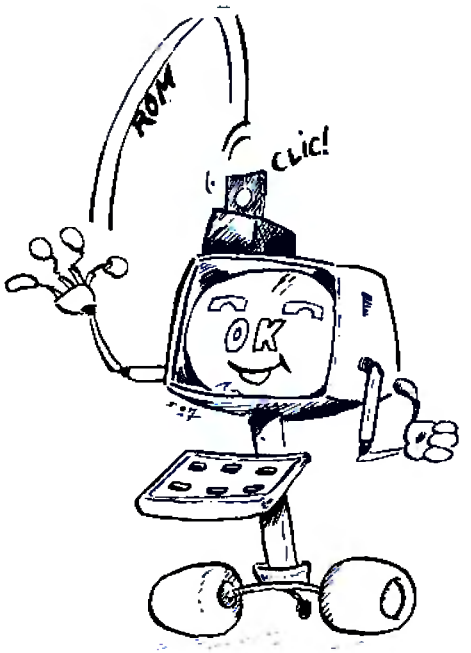
Annexe C — FORTH, c'est aussi bien d'autres choses 140
C.1 FORTH = souplesse + rapidité 140
C.2 Une idée du fonctionnement de FORTH 144

Annexe D — Codes ASCII 149

Annexe E — Messages d'erreur 153

Annexe F — Solutions des exercices 156

Introduction



Ce premier manuel est un manuel d'initiation au langage FORTH, et n'a pas la prétention d'être exhaustif. Lorsque vous aurez besoin d'un détail supplémentaire, vous pourrez consulter le MANUEL DE RÉFÉRENCE, qui décrit commande par commande le fonctionnement du FORTH.

Ce FORTH contient le Standard FORTH-79, considérablement augmenté pour tenir compte des possibilités graphiques et musicales des machines TO 7, TO 7-70 ou MO5.

Il n'est pas possible d'apprendre un langage sans faire de nombreux exercices. Nous vous invitons à étudier ce manuel progressivement, à essayer tous les exercices, et à les taper au clavier.

N'hésitez pas à rechercher des solutions personnelles, en travaillant constamment avec le MANUEL DE RÉFÉRENCE. Plus vous le consulterez, plus les commandes et la logique du langage vous seront familières (!!!).

Les possibilités du FORTH sont bien plus étendues que celles que nous décrivons ici, et votre ordinateur est prêt à les mettre en œuvre. Malgré cela, il s'agit d'un langage très simple à utiliser, et de surcroît très performant (Que demander de plus ?).

Nous espérons que vous prendrez beaucoup de plaisir à pratiquer ce langage, et que vous ferez bientôt partie des passionnés du FORTH.

Cela dit, au travail !

taper son nom au clavier. Nous parlerons de MOTS, sans faire de distinction entre les actions et leurs noms.

Par exemple, tapons BELL, ce qui se fait en tapant les touches [B], [E], [L], [L], et la touche marquée [ENTRÉE]. L'ordinateur émet un son...

Le mot dont le nom est BELL correspond à l'action « faire un bip ». Pour le distinguer du bip émis lorsqu'on tape une touche, le son réalisé par BELL est un accord musical. Aussitôt après, l'ordinateur affiche OK pour nous dire :

« J'ai fini. J'attends de nouvelles instructions ».

Lorsqu'il n'affiche pas de OK, c'est qu'il est en train de travailler, ou que vous avez commencé à taper un ordre, mais que vous n'avez pas encore tapé la touche [ENTRÉE].

C'est une habitude à prendre, lorsque vous voulez confirmer votre entrée, vous tapez la touche [ENTRÉE]. Tant que cela n'est pas fait, vous pouvez faire marche arrière sur la même ligne, en utilisant la touche [EFF].

Les autres touches de déplacement de curseur fonctionnent, mais ne sont pas comprises par FORTH (du moins pas en tant que touches de déplacement).

On peut donner plusieurs ordres à la fois, en les séparant par des espaces.

BELL BELL BELL [ENTRÉE] OK

L'ordinateur a répondu OK après avoir fait trois sons.

Les espaces servent de **séparateurs**, c'est-à-dire qu'ils servent au FORTH à distinguer dans ce que nous tapons (On parle en FORTH du « flot d'entrée ») quels mots il va devoir exécuter. Ils sont donc **fondamentaux** ; si on les oublie, l'ordinateur comprendra qu'il doit exécuter le mot BELLBELLBELL, qu'il ne connaît pas. C'est pourquoi lorsque nous tapons BELLBELLBELL, (suivi de ENTRÉE), il nous répond :

BELLBELLBELL [ENTRÉE] ? UK ERROR (UnKnown, c'est-à-dire nom inconnu).

Pour faire disparaître ce message de l'écran, tapez CLS.

C'est un autre mot, qui efface l'écran.

1-2 Le dictionnaire

Il est donc important de connaître les mots dont on dispose. Nous pouvons en afficher la liste en tapant :

VLIST C'est un mot comme un autre !!.

Nous pouvons à tout moment arrêter ou reprendre le défilement en frappant une touche quelconque. Si à ce moment vous frappez la touche STOP, le défilement s'arrête définitivement et vous reprenez la main au clavier, pour une autre commande (par exemple VLIST...).

Comme vous pouvez le voir, les mots FORTH ne sont pas nécessairement écrits avec les lettres de l'alphabet. Tous les symboles accessibles au clavier peuvent être utilisés, sauf bien entendu les espaces, qui servent à séparer les mots...

Par exemple &"$$ §!d%9 F>**/) serait un nom valide, ce qui ne veut pas dire qu'il correspond pour le moment à une action.

Conformément au standard FORTH-79, les noms des mots du FORTH peuvent comporter jusqu'à 31 caractères.

A propos du clavier :

N'ayez pas peur de faire des essais, vous ne pouvez pas faire de mal à votre ordinateur. Le plus probable est que vous receviez le message ? UK ERROR, mais cela est sans importance. Si par hasard vous perdez la main en essayant de taper des mots du dictionnaire (par exemple si vous tapez >R suivi de la touche ENTRÉE), vous pouvez toujours éteindre et rallumer votre unité centrale, ou appuyer sur le bouton d'initialisation programme.

— Les touches particulières (les flèches, les touches [INS], [EFF], [CNT], [RAZ], [STOP], [ACC]) correspondent à des caractères spéciaux. Elles peuvent être utilisées dans les noms... Par exemple, si on tape [B], [E], [L], [L], [→], [B], [E], [L], [L] l'ordinateur considérera cela comme UN mot de 9 lettres (La flèche à droite n'est pas un espace).
— Par contre, on ne peut pas utiliser de caractères accentués pour les noms des mots FORTH. Les caractères minuscules ne posent pas de problème (Un A n'est pas la même chose qu'un a), mais les caractères accentués ne sont pas vraiment des caractères, ce sont des suites de plusieurs caractères... (voir ci-dessous).
— Les caractères écrits en jaune sur les touches sont obtenus en appuyant simultanément sur la touche concernée et sur une des deux touches [.] (majuscules).
— On peut aussi passer de l'écriture « majuscules » en « minuscules » (et vice versa) en appuyant simultanément sur la touche « espace » et la touche [.]. Par exemple, si on tape [ ] [.] puis les touches [A], [B], [C], l'ordinateur comprendra qu'on a tapé « abc ». Si alors on retape les mêmes touches, il écrira « ABC » (repasse en mode « majuscules »).
— Les voyelles minuscules accentuées s'obtiennent par des séquences de trois touches : la touche [ACC], une touche accent, et la voyelle minuscule à accentuer.

Par exemple, pour obtenir un « é », il faudra taper successivement (après avoir vérifié qu'on est bien en mode minuscules) la touche [ACC], puis simultanément les touches [.] et 7 (pour obtenir un accent aigu), et enfin la touche [E].
— Le « ç » s'obtient par la séquence [ACC] [c] [c] (en mode « minuscules »)
— En outre, on obtiendra les crochets (qui sont des caractères importants en FORTH) en tapant simultanément [CNT] et [A] (] ou [CNT] et [Z] (]).

1-3 Définir des nouveaux mots

Une des particularités les plus intéressantes de FORTH est que c'est un langage extensible, en ce sens que nous pouvons ajouter des nouveaux mots (les nôtres) au dictionnaire. Pour définir un nouveau mot, par exemple un mot qui fasse 2 bips, efface l'écran, puis fasse encore 3 bips, il faut d'abord lui choisir un nom : 2BIPS-CLS-3BIPS (Pourquoi pas ?)
La suite est facile : Tapons
: 2BIPS-CLS-3BIPS BELL BELL CLS BELL BELL BELL ; ENTRÉE
Votre ordinateur doit vous répondre simplement OK. Sinon, vérifiez que vous avez bien recopié exactement la ligne, **en respectant les espaces.**
Si vous n'avez aucune réponse, c'est que vous n'avez pas terminé (avez-vous bien mis le point-virgule ?)
Si la réponse est UK ERROR (UnKnown), il indique le mot qu'il n'a pas compris. Les erreurs les plus fréquentes sont les mots mal orthographiés (Attention, l'ordinateur est intraitable à ce point de vue), ou des espaces manquants. Retapez éventuellement la ligne (sans utiliser les touches spéciales, sauf la touche EFF, si vous avez fait une erreur), et ne vous souciez pas du message NU (Not Unique) qui apparaîtra.

Explications :

Le mot : (deux points) est un mot comme un autre, qui envoie à l'ordinateur le message suivant :
« L'utilisateur va définir un nouveau mot, dont le **nom** est le prochain nom du flot d'entrée (ici 2BIPS-CLS-3BIPS), et dont **l'action** consistera en l'exécution des mots suivants, jusqu'au mot ; (point-virgule) ».
Il est donc indispensable de laisser un (ou plusieurs) espaces après les deux-points, et de même avant le point-virgule.

Voilà, cela n'est pas plus compliqué que cela. Notre dictionnaire s'est enrichi d'un nouveau mot, comme nous pouvons le vérifier en faisant VLIST, et en arrêtant le défilement presque aussitôt. Notre mot, étant le dernier créé, est le premier lu (Nous aurons souvent à prendre en compte ce genre de phénomène : dernier arrivé, premier parti...).
Vérifions aussi qu'il fonctionne bien comme prévu, il suffit de taper son nom (suivi bien entendu de la touche ENTRÉE)... Correctement orthographié, bien entendu.
Bien sûr, on choisit en général des noms plus courts, mais il ne faut pas pour autant que les mots n'aient plus de signification... Il n'est pas toujours facile de trouver des noms qui soient à la fois courts et parlants. Dans ces situations, l'habitude est de donner la priorité à la clarté. Le FORTH de base comporte déjà suffisamment d'abréviations mystérieuses (mais standardisées), nous le verrons bientôt...

Comme on vient de le voir, FORTH a deux modes de fonctionnement :
— Le mode **exécution** ; lorsqu'on tape des noms au clavier pour que les mots correspondants soient exécutés

— Le mode **définition** (Les spécialistes du FORTH parlent souvent de mode **compilation**), lorsqu'on tape les noms entre : et ;. Dans ce cas, on définit des nouveaux mots, qui sont mis en réserve en haut du dictionnaire, en vue d'une utilisation ultérieure (pour une exécution, ou pour définir d'autres mots). Et les mots : et ; ont seulement pour action de passer d'un mode à l'autre.

En fait, la cartouche contient un programme qui s'exécute tant que votre ordinateur est sous tension. Cette exécution consiste à interpréter en FORTH ce que vous lui donnez à faire (le flot d'entrée), et à le traiter. Le traitement pouvant consister soit en une mise au dictionnaire, soit en une exécution (au sens du FORTH).
Si les mots étrangers vous gênent, vous pouvez très bien vous faire votre FORTH à vous, en renommant simplement les commandes...

Exemple :
: DRELIN BELL ; : ECRAN-BLANC CLS ; OK
Dorénavant, lorsque vous taperez DRELIN, vous entendrez un bip...

1-4 Quelques mots simples

Pour construire des exemples, il nous faut disposer d'un vocabulaire minimum. Voici donc quelques mots, concernant surtout l'affichage.

Pour afficher un message à l'écran, par exemple : COUCOU, C'EST MOI la commande FORTH est :
." COUCOU, C'EST MOI"
." Le mot ." (point-guillemet) est un mot du FORTH, donc doit être suivi d'un espace. Cet espace ne fait pas partie du message qui suit (La première lettre du message est bien le C de COUCOU). Le deuxième guillemet marque la fin du message, mais n'est pas un mot du FORTH. Si on écrit un espace après le I de MOI, cet espace fera partie du message.

Quelques autres mots...

CR	Va à la ligne (Même effet que la touche ENTRÉE, mais dans une définition).
SPACE	Affiche un espace.
x **SPACES**	Affiche x espaces (5 SPACES affichera 5 espaces).
CLS	Vide l'écran du téléviseur, et replace le curseur en haut à gauche.
INVCOLOR	Inverse les couleurs du fond et des caractères pour les affichages à venir.

Donnons un ou deux exemples de réalisations en FORTH :

Exemple 1 : (Respectez les blancs, et passez à la ligne par la touche ENTRÉE).
```
: EXEMPLE BELL BELL CLS CR CR 11 SPACES
          ." Un nouveau mot : il s'appelle "
          INVCOLOR ." EXEMPLE " INVCOLOR .
          CR CR BELL BELL ;
```

Ce que fait ce mot est facile à comprendre. A l'intérieur même de la définition de EXEMPLE, on peut utiliser la touche ENTRÉE pour mieux séparer les groupes d'actions (Préparer l'affichage, afficher la première partie, puis la deuxième avec des couleurs inversées, et arrêter sur deux bips, après avoir sauté deux lignes), sans que cela influe sur l'exécution. Les deux parties du message ont été entrées sur deux lignes différentes, mais seront affichées sur la même ligne... En mode **définition**, il est nécessaire d'utiliser le mot CR pour obtenir un passage à la ligne lors de l'**exécution** ultérieure du mot qu'on est en train de définir.

Exemple 2 :

Nous désirons écrire de grandes lettres à l'écran. Un des moyens (il y en a de plus compétitifs...) est de commencer par définir des « briques », que nous pourrons simplement empiler comme dans un jeu de construction.

```
: PAVE INVCOLOR ."  " INVCOLOR ;
: BARRE INVCOLOR ."      " INVCOLOR ;
```

PAVE affiche un espace en vidéo inverse, BARRE en affiche 5. (Les guillemets de PAVE doivent contenir 2 espaces, ceux de BARRE doivent en contenir 6). Nous pouvons tester ces deux mots en tapant leurs noms...

Notons aussi que nous aurions pu aussi définir BARRE par :

```
: BARRE PAVE PAVE PAVE PAVE PAVE ;
```

Puis par exemple :

: E CR BARRE CR PAVE CR BARRE CR PAVE CR BARRE CR ; écrira un E en grands caractères lorsqu'on tapera la commande E... Ou encore, pour faire du morse :

```
: TRAIT BARRE SPACE ;
```

et :

```
: POINT PAVE SPACE ;
```

(Ne nous occupons pas du message POINT NU, nous y reviendrons au paragraphe 1-6)

Nous avons maintenant tout ce qu'il nous faut... Essayons :

```
TRAIT POINT POINT TRAIT POINT POINT POINT TRAIT ...
```

Exercice :

1.a - Faire un mot qui affiche FORTH en grand sur l'écran...

1-5 Supprimer des mots du dictionnaire

Bien sûr, si on peut augmenter le dictionnaire, on doit aussi pouvoir le réduire. (Vous ne serez pas toujours satisfait des mots que vous avez créés... ou encore, vous voudrez certainement essayer des mots, de manière provisoire, ne fut-ce que pour vous entraîner...).

Il y a plusieurs moyens de supprimer des mots du dictionnaire.

Le plus radical est d'arrêter l'unité centrale. Tous les mots que vous aurez fabriqués disparaissent, il ne reste que ceux de la cartouche, qui constituent le dictionnaire de base.
Le mot COLD a le même effet, mais est moins brutal.

Un moyen plus paisible consiste à utiliser le mot FORGET. Par exemple, définissons plusieurs mots au hasard...

```
: XX CR ;  OK          Rebaptise simplement la commande CR...
: YY ;  OK             Ne fait rien...
```

La frappe de VLIST nous prouve que XX et YY sont bien en haut du dictionnaire. Si maintenant nous tapons FORGET YY, ce dernier mot sera supprimé. Vérification :

```
YY  ? UK ERROR         Bien sûr !!!
```

Mais il y a plus curieux... Tapons par exemple :

```
: YY ; OK
YY  OK                 Pas de problème, l'ordinateur n'a rien fait...
FORGET XX  OK          Oublions le mot XX...
YY  ? UK ERROR         Que se passe-t-il ? Nous n'avons pas demandé d'oublier YY !...
```

VLIST pour vérifier : c'est bien vrai, YY n'est plus là.

L'explication est la suivante : tout se passe comme si les mots du dictionnaire étaient empilés comme une pile d'assiettes. Lorsqu'on veut supprimer le deuxième, celui du dessus disparaît en même temps... (et lorsqu'on en rajoute un, il vient se placer sur le dessus).

Vous pouvez vous amuser à créer 5 ou 6 mots, et à en oublier pour bien comprendre ce fonctionnement. Mais bien entendu, il est hors de question d'oublier les mots de la cartouche.

Un conseil : Lorsque vous commencez à écrire des mots d'essai, que vous n'avez pas l'intention de garder, commencez par taper un mot vide, qui ne fasse rien... Par exemple : XX ; Pour oublier vos essais, vous n'aurez ensuite qu'à taper FORGET XX.

1-6 Mots en plusieurs exemplaires

FORTH est très complaisant, et ne vous refusera presque jamais rien. Par exemple, si vous essayez de redéfinir un mot, il n'y verra pas d'inconvénient. Tout au plus vous donnera-t-il un avertissement.

Exemple :

```
: CR ." NON, JE N'IRAI PAS A LA LIGNE" ; CR NU  OK
```

Et maintenant :

```
CR NON, JE N'IRAI PAS A LA LIGNE  OK
```

Une fois de plus, les derniers ont raison... C'est la dernière définition d'un mot qui est prise en compte, lors de son exécution, ou lors de la création d'un autre

— Le mode **définition** (Les spécialistes du FORTH parlent souvent de mode **compilation**), lorsqu'on tape les noms entre : et ;. Dans ce cas, on définit des nouveaux mots, qui sont mis en réserve en haut du dictionnaire, en vue d'une utilisation ultérieure (pour une exécution, ou pour définir d'autres mots). Et les mots : et ; ont seulement pour action de passer d'un mode à l'autre.

En fait, la cartouche contient un programme qui s'exécute tant que votre ordinateur est sous tension. Cette exécution consiste à interpréter en FORTH ce que vous lui donnez à faire (le flot d'entrée), et à le traiter. Le traitement pouvant consister soit en une mise au dictionnaire, soit en une exécution (au sens du FORTH).
Si les mots étrangers vous gênent, vous pouvez très bien vous faire votre FORTH à vous, en renommant simplement les commandes...

Exemple :
: DRELIN BELL ; : ECRAN-BLANC CLS ; OK
Dorénavant, lorsque vous taperez DRELIN, vous entendrez un bip...

1-4 Quelques mots simples

Pour construire des exemples, il nous faut disposer d'un vocabulaire minimum. Voici donc quelques mots, concernant surtout l'affichage.

Pour afficher un message à l'écran, par exemple : COUCOU, C'EST MOI la commande FORTH est :
." COUCOU, C'EST MOI"

." Le mot ." (point-guillemet) est un mot du FORTH, donc doit être suivi d'un espace. Cet espace ne fait pas partie du message qui suit (La première lettre du message est bien le C de COUCOU). Le deuxième guillemet marque la fin du message, mais n'est pas un mot du FORTH. Si on écrit un espace après le I de MOI, cet espace fera partie du message.

Quelques autres mots...

	CR	Va à la ligne (Même effet que la touche ENTRÉE, mais dans une définition).
	SPACE	Affiche un espace.
x	**SPACES**	Affiche x espaces (5 SPACES affichera 5 espaces).
	CLS	Vide l'écran du téléviseur, et replace le curseur en haut à gauche.
	INVCOLOR	Inverse les couleurs du fond et des caractères pour les affichages à venir.

Donnons un ou deux exemples de réalisations en FORTH :

Exemple 1 : (Respectez les blancs, et passez à la ligne par la touche ENTRÉE).

```
: EXEMPLE BELL BELL CLS CR CR 11 SPACES
          ." Un nouveau mot : il s'appelle "
          INVCOLOR ." EXEMPLE " INVCOLOR
          CR CR BELL BELL ;
```

l'éditeur sont rangées dans une zone de la mémoire de l'ordinateur. On peut les mettre au dictionnaire, ou les modifier, ou encore les sauvegarder sur cassette ou sur disquette (pour pouvoir les remodifier ultérieurement).
L'étude des commandes de l'éditeur ne fait pas partie de l'étude du FORTH, mais elle n'est pas très longue, et ne présente pas de difficulté. En contrepartie de ce petit effort, l'utilisation de l'éditeur facilitera beaucoup vos essais et vos corrections.

Dans un premier temps, le mieux est probablement d'apprendre à utiliser les principales commandes (Annexe A, paragraphes A-1 et A-2-1), et d'attendre d'avoir des applications de taille plus importante pour continuer l'étude de l'éditeur.

1-7-2 Les périphériques

Si votre « application » (il est bien difficile, en FORTH, de parler de programmes) vous plaît, si vous désirez conserver votre dictionnaire et le retrouver dans cet état une autre fois, vous pouvez la sauvegarder sur une cassette magnétique, ou sur disquette.
Si vous avez une imprimante, vous voulez peut-être obtenir vos affichages sur papier...

Vous pouvez dans ce cas étudier l'annexe B, relative aux périphériques, qui vous expliquera comment procéder.

Cette étude n'est toutefois pas très urgente ; il est préférable d'avoir auparavant fait un bon tour d'horizon des possibilités du langage. D'ailleurs, l'éditeur offre déjà beaucoup de possibilités dans ce domaine.

Chapitre 2

Une pile de nombres

On peut travailler en FORTH avec plusieurs sortes de nombres. De base, les nombres standards sont les nombres entiers compris entre − 32768 et + 32767...
Peut-être cela vous semble-t-il être une limitation, mais c'est dans l'esprit du langage de travailler avec des objets les plus simples possibles, pour aller plus vite. De plus, FORTH dispose d'une très grande richesse au point de vue fonctions et opérations, qui compense dans une large mesure cette limitation.

Le chapitre 8 fait le point sur les différents types de nombres. Nous ne nous intéresserons pour le moment qu'aux nombres « standards ».

2-1 Écritures et affichage des nombres

L'ordre d'affichage d'un nombre est le **point** (.). C'est un drôle de nom pour une commande, mais pourquoi pas ?

Exemple :

```
345 .       345  OK
12 . 13 . 14 .  12 13 14  OK
```

Le point affiche un nombre, suivi d'un espace (pour des raisons évidentes de lisibilité).
Attention, il s'agit d'une commande FORTH. C'est-à-dire qu'elle doit absolument être **précédée et suivie d'un espace.**

La commande 345. aurait une autre signification pour l'ordinateur (chapitre 8)... De manière plus générale, nous vous demanderons de ne pas taper de point dans les nombres, cela ne serait pas compris comme un nombre décimal (Patience, l'explication viendra en son temps ; il vous suffit pour le moment de vous souvenir que FORTH ne connaît que des nombres entiers).
D'autre part, les nombres positifs s'écrivent sans le signe + . Si vous tapez +45, vous aurez droit une fois de plus au sempiternel UK ERROR. Cela n'est pas une limitation, et laisse +45 disponible pour un nom de mot FORTH (par exemple « ajouter 45 »).
Les nombres négatifs sont bien sûr représentés avec le simple « − », comme d'habitude. L'écriture −10 ne pose pas de problème.

Par ailleurs, le point s'écrit **après** le nombre qu'il doit afficher... FORTH procède en disant : « Je prends le nombre 345, et je l'affiche » (Le nombre d'abord, l'utilisation ensuite).
Il s'agit là d'une particularité fondamentale du FORTH, que les spécialistes appellent la **notation postfixée**, ou **notation polonaise inversée**. Tous les mots FORTH, sauf exception, fonctionneront ainsi...

Nous avons déjà rencontré ce phénomène au premier chapitre, avec le mot SPACES. Pour afficher 11 espaces, nous avons tapé la commande 11 SPACES (SPACES utilise le nombre 11).

2-2 La pile des données

L'explication de ce qui précède est la suivante : lorsqu'un mot FORTH a besoin d'un nombre, il va toujours le chercher à un endroit de la mémoire réservé à cet effet. Et lorsqu'on tape un nombre au clavier, c'est à cet endroit qu'il va se ranger.
Cette zone de la mémoire est le cœur du FORTH, c'est la **pile des données.**

Les mots FORTH uilisent le (ou les) nombre(s) du dessus de la pile, et lorsqu'on tape un nombre, FORTH exécute l'action « Le mettre sur la pile » (Tout se passe comme si les nombres étaient des mots FORTH particuliers).

```
12 OK    La pile contient le nombre 12.
13 OK    La pile contient 12 et 13 (13 au-dessus de 12)
. 13 OK  Affiche le nombre qui est au sommet de la pile.
```

Ou encore, en une seule entrée :

```
12 13 . . 13 12 OK
```

Attention au mécanisme, c'est une fois de plus le dernier entré qui est le premier utilisé, de sorte que l'ordre d'affichage est inverse de l'ordre des entrées.

```
1 2 3 4 5 6 . . . . . 6 5 4 3 2 1 OK
```

On peut à tout moment visualiser le contenu de la pile des données grâce au mot .S (Le point pour rappeler qu'il s'agit d'afficher des nombres, et le S parce qu'en anglais, le mot « pile » se traduit par « stack »)

```
1 2 3 .S donne :
PARAMETER : 1 2 3
RETURN :
```

La pile des données s'appelle en anglais « parameter stack » (pile des paramètres). Nous ne nous occuperons pas pour le moment de la pile des retours (« return stack »). Le sommet de la pile est le nombre le plus à droite (ici le nombre 3). Si nous tapons maintenant un point :

```
. 3 OK
.S donnera :
PARAMETER : 1 2 ...
. " SALUT" SPACES ." BONJOUR" SALUT  BONJOUR  OK
```

Entre SALUT et BONJOUR, il y a **deux** espaces
(SPACES a utilisé le 2 qui était en haut de la pile)

```
.S donne maintenant : PARAMETER : 1 ...
. 1 OK normal...
. 0 ? ES ERROR (Empty Stack) La pile était vide !!!
```

2-3 Opérations arithmétiques fondamentales

Ouf !... La notation est la même que d'habitude : on les note respectivement + (addition), − (soustraction), * (multiplication), et / (division). Mais bien sûr, ces actions respectent la même logique que les autres mots FORTH. De sorte que lorsqu'on tape + , par exemple, cela suppose que les deux nombres à additionner sont déjà sur la pile des données. Autrement dit, pour additionner les nombres 12 et 54, on tapera :

```
12 54 + 12 dépose sur la pile le nombre 12
        54 dépose sur la pile le nombre 54
         + effectue l'addition
```

Où est le résultat ? Si on veut pouvoir le réutiliser (pour affichage, ou pour un autre calcul) où peut-on bien le mettre ? Sur la pile, bien entendu !
Par conséquent, si nous tapons un point, nous devrions voir le résultat :

```
66  OK Ça marche !!!
```

Les autres opérations procèdent de la même manière. Il faut toutefois noter que la division est la division entière (puisque les nombres sont entiers), c'est-à-dire que si par exemple on divise 17 par 5, le résultat sera 3 (Car 17 = 3 × 5 + 2 ... Ou encore, si vous préférez, le résultat avec une virgule serait 3,4 et il est tronqué à sa partie entière, 3).

Exemples :

```
2 3 + . 5  OK
2 3 * . 6  OK
2 3 - . -1  OK
2 3 / . 0  OK
17 5 / . 3  OK
```

FORTH possède aussi deux autres opérations fondamentales, MOD et /MOD, qui complètent la division.
MOD laisse sur la pile le reste de la division (sans le quotient).
/MOD laisse sur la pile le reste et le quotient (le quotient est au sommet de la pile).

Exemple :

```
17 5 MOD . 2  OK
17 5 /MOD . . 3  2  OK
```

Le quotient est au-dessus, il est donc le premier affiché.

Il n'est bien sûr pas nécessaire de réécrire les nombres à chaque fois, on peut utiliser le résultat qui est sur la pile. Si vous voulez calculer la somme des quatre nombres 23, 34, 45 et 56, on peut utiliser indifféremment :

```
23 34 + 45 + 56 + . 158  OK
```

ou

```
23 34 45 56 + + + . 158  OK
```

ATTENTION : Les nombres utilisés sont les nombres entiers entre −32768 et 32767.

Exemple :

```
100 1000 * . −31072  OK
```

Le résultat est évidemment faux !!! La « philosophie » de FORTH est de **ne faire aucune vérification de la validité de ses actions. A vous de prendre les précautions nécessaires !!!**
Cela n'a pas que des inconvénients : d'une part cela accélère nettement l'exécution, d'autre part cela laisse à l'utilisateur la possibilité de gérer lui-même les situations d'erreur.

2-4 Notation conventionnelle des actions sur la pile

Il est bien sûr très important de connaître l'effet des mots FORTH sur la pile des données. La convention habituellement utilisée est la suivante :
L'action du mot est symbolisée par 3 traits. A gauche, on écrit les nombres de la pile avant exécution, et à droite ce que laisse le mot après son exécution. Pour ces deux notations, le sommet de la pile est le nombre le plus à droite.

Exemple :

+	n1 n2 --- n	n est la somme de n1 et n2
−	n1 n2 --- n	n = n1 − n2 (soustraction)
*	n1 n2 --- n	n = produit de n1 par n2
/	n1 n2 --- n	n = quotient entier de n1 par n2
MOD	n1 n2 --- n	n = reste de la division de n1 par n2
/MOD	n1 n2 --- n3 n4	Effectue la division de n1 par n2. n4 est le quotient, n3 est le reste.

On n'écrit bien entendu que les nombres qui sont modifiés par le mot.

Exemple :

1 2 3 4 + OK

Avant l'exécution de +, la pile contient 1, 2, 3 et 4 (sommet). Après l'addition, elle contient 1, 2, et 7 (les paramètres de l'addition sont les deux nombres du dessus de la pile). 1 et 2 ne sont pas modifiés, on n'en parle donc pas. Le mot + a simplement **remplacé** 3 et 4 par 7.

De la même manière, nous pourrions écrire :

BELL	---	Emet un bip
CLS	---	Efface l'écran vidéo
SPACE	---	Écrit un espace
SPACES	n ---	Écrit n espaces
INVCOLOR	---	Inverse les couleurs fond-caractère
	n ---	Affiche le nombre n

Nous utiliserons désormais cette notation pour décrire les mots que nous allons étudier ou décrire.

2-5 Manipulations sur la pile

Faire des calculs en FORTH, c'est la plupart du temps travailler avec la pile, en transformant les données que nous y avons déposées.
Mais comme rien (ni personne) n'est parfait, ces données ne sont pas toujours dans le bon ordre, ou parfois on n'a plus besoin d'un résultat antérieur... Il faut donc parfois « réarranger » la pile à notre convenance.
Eh bien, C'EST PRÉVU ! Les manipulations de pile sont simples à décrire, un peu plus délicates à mémoriser, pas toujours faciles à utiliser de manière optimale.

Nous allons énumérer les principales avant de voir les exemples.

DUP	n --- n n	duplique le sommet de la pile
DROP	n ---	supprime le sommet de la pile
SWAP	n1 n2 --- n2 n1	échange les deux nombres au sommet de la pile
OVER	n1 n2 --- n1 n2 n1	recopie n1 (le deuxième nombre à partir du sommet) au-dessus de la pile.
ROT	n1 n2 n3 --- n2 n3 n1	ramène le troisième nombre au-dessus de la pile. Les mathématiciens parlent de permutation circulaire (n1 devient n2, n2 devient n3, et n3 devient n1, on tourne en rond...)

Toutes ces opérations existent en « modèle familial », on peut aussi manipuler la pile par paquets de 2 nombres. Il suffit d'ajouter le préfixe « D » aux mots correspondants.

DDUP	n1 n2 --- n1 n2 n1 n2	duplique « en bloc » les deux nombres du dessus de la pile.
DDROP	n1 n2 ---	supprime deux nombres au sommet de la pile
DSWAP	n1 n2 n3 n4 --- n3 n4 n1 n2	travaille comme SWAP, échange (n1 n2) et (n3 n4)
DOVER	n1 n2 n3 n4 --- n1 n2 n3 n4 n1 n2	comme OVER, recopie (n1 n2) au-dessus de la pile
DROT	n1 n2 n3 n4 n5 n6 --- n3 n4 n5 n6 n1 n2	le plus délicat... Ramène (n1 n2) au sommet de la pile ; (n3 n4) et (n5 n6) descendent de 2 positions.

Il y en a d'autres, mais celle-ci sont les plus utilisées.

Un peu d'entraînement :

Exercices :

2.a - Essayez de prévoir le résultat avant de taper la touche ENTRÉE ...

1 2 3 4 5 SWAP .S ENTRÉE
1 2 3 4 5 DUP .S ENTRÉE
1 2 3 4 5 ROT .S ENTRÉE
1 2 3 4 5 DROP .S ENTRÉE
1 2 3 4 5 OVER .S ENTRÉE

Et ainsi de suite... Vous pouvez essayer vos propres séquences, jusqu'au moment où au moins ces cinq commandes vous seront bien familières ; les autres sont moins fréquentes.

2.b - On veut faire un mot qui affiche le carré d'un nombre : Il s'agit de le multiplier par lui-même, nous devons donc l'avoir en deux exemplaires. D'où le mot :
: CARRE DUP * . ; 17 CARRE 289 OK (Le saviez-vous ?)
Faire un mot CUBE, qui donnera le cube d'un nombre. Quel est le plus grand nombre pour lequel le résultat sera correct ?
2.c - Quelle différence y a-t-il entre DDUP et DUP DUP
2.d - Que fait le mot X défini par : X OVER OVER ;
2.e - Même question pour le mot Y défini par : Y ROT ROT ;

2-6 Opérations en chaîne

Nous allons passer à des choses plus complexes, et mélanger un peu les opérations et les manipulations de pile. Mais revenons d'abord sur les opérations. La notation postfixée est peut-être peu commode au premier abord, elle n'en est pas pour autant sans intérêt, loin de là. D'ailleurs, certaines calculatrices de poche ont adopté ce système en raison de sa maniabilité...

Par exemple, si on veut calculer a = (2 + 5) × (3 + 6)...
Pour afficher a, le plus simple sera de calculer ainsi :
2 5 + 3 6 + * . 63 OK

La partie 2 5 + calcule la somme, et dépose le résultat sur la pile. 3 6 + fait de même. Après 2 5 + 3 6 +, la pile contient donc les nombres suivants : 7 9 ---
Après multiplication, elle contient donc le nombre 63.

Nous n'avons utilisé **aucune parenthèse, aucune priorité dans l'exécution des actions**, nous n'avons eu à prendre en compte que la **position des nombres sur la pile**.

Essayons maintenant de calculer b = (23 × 11) + (34 × 17)

Nous avons à calculer les deux produits, puis à ajouter les résultats.
23 11 * calcule le premier produit, et laisse le résultat sur la pile.
34 17 viennent se mettre au-dessus.
L'état de la pile est : 253 34 17 ---
* calcule le produit 34 × 17, et laisse le résultat sur la pile.
L'état de la pile est : 253 578 ---
+ Fait l'addition. La pile devient : 831 ---

Il ne reste plus qu'à afficher le résultat, ce qui se fait par le mot « point ».

D'où la séquence :
23 11 * 34 17 * + . 831 OK

Notez la parfaite ressemblance avec l'exemple précédent. C'est un des avantages de la notation postfixée, addition et multiplication ont le même statut (alors que dans la notation algébrique traditionnelle, la multiplication est prioritaire sur l'addition).

Tout cela est fort bien, mais que faire si nos quatre nombres sont déjà sur la pile, qu'ils soient les résultats de calculs antérieurs, ou que nous désirions faire un mot (qui calcule ab + cd lorsqu'on lui donne a, b, c, d).
L'état initial de notre pile est donc :
a b c d ---

Nous pouvons effectuer le produit cd, puisque c et d sont sur la pile.
* a b c d --- a b cd

Il nous faut maintenant amener a et b sur le dessus pour faire le second produit. Nous avons plusieurs moyens pour cela :

Entre autres :
SWAP ROT a b cd --- a cd b --- cd b a
SWAP ROT

ou encore :
ROT ROT a b cd --- b cd a --- cd a b
ROT ROT

Il ne reste qu'à faire le produit et à additionner.
D'où le mot ab+cd (sans espace, c'est un mot FORTH), défini comme suit :
: ab+cd * SWAP ROT * + ;

On peut maintenant taper :
23 11 34 17 ab+cd . 831 OK

Notez au passage comme il est agréable de pouvoir utiliser des mots avec des noms aussi parlants que « ab+cd »... C'est un des grands avantages du FORTH...

Exercices :

Construire des mots qui calculent les expressions suivantes :
2.f- a b c d --- x où x = ab − cd
2.g- a b c d --- x où x = ad − bc
2.h- a b --- x où x = (a − b)2 (carré du nombre (a − b))
2.i- a b --- x où x = a2 − b2 (différence des deux carrés)
2.j- x --- y où y = 2×2 + 3x − 5

2.k- A propos de /MOD...
125 60 /MOD . . 2 5 OK (125 = 2×60 + 5)
Autrement dit, 125 secondes font 2 minutes et 5 secondes.
Faire un mot HMS qui s'exécute sous la forme × HMS et qui réponde les heures, minutes, secondes. Par exemple on devra pouvoir taper : 7273 HMS
2 h. 1 mn. 13 s. OK

2-7 Autres opérations

FORTH est très riche en opérations. Il s'agit en fait d'opérateurs plutôt que d'opérations, en ce sens que ce sont des ACTIONS qui modifient l'état de la pile.

NEGATE	n1 --- n2	n2 est l'opposé de n1 (change le signe)
ABS	n1 --- n2	n2 est la valeur absolue de n1.
MAX	n1 n2 --- n3	n3 est le plus grand des deux nombres n1 et n2.
MIN	n1 n2 --- n3	n3 est le plus petit des deux nombres n1 et n2.

Il y en a encore d'autres, que vous découvrirez dans le Manuel de Référence, au fur et à mesure de vos investigations...

```
-45 NEGATE . 45  OK
-45 ABS . 45  OK
45 NEGATE . -45  OK
45 ABS . 45  OK
1234 2345 MAX . 2345  OK
1234 2345 MIN . 1234  OK
-3 -4 MAX . -3  OK (C'est bien vrai !)
-3 -4 MIN . -4  OK (idem)
```

MAX et MIN sont deux mots très pratiques dans certains cas. Supposons par exemple que vous vouliez que le nombre sur la pile soit compris entre Ø et 12. S'il est inférieur à Ø, on prendra Ø, et s'il est supérieur à 12, on prendra 12. Autrement dit, notre résultat doit d'abord être un nombre positif ou nul. Cela se fait en prenant le MAX du nombre et de Ø.
Il reste sur la pile un nombre positif ou nul, mais qui peut être supérieur à 12. Il suffit de prendre maintenant le MIN de ce nombre et de 12, le résultat sera dans les bonnes limites.
D'où le mot :

```
: (Ø-12) Ø MAX 12 MIN ;
```

Puis :

```
5 (Ø-12) . 5  OK
25 (Ø-12) . 12  OK
-3 (Ø-12) . Ø  OK
Ø (Ø-12) . Ø  OK
```

2-8 FORTH et la rapidité

Si vous avez regardé de près la liste des mots du FORTH, vous avez peut-être été surpris d'y trouver des mots inattendus, comme 0, 1, 2, 3... En effet, il s'agit bien là de mots FORTH comme les autres. Leur raison d'être est la suivante :
Lorsque FORTH rencontre un mot à exécuter, la première chose qu'il fait est d'aller voir dans le dictionnaire si ce mot est connu. Si oui, il l'exécute, sinon il essaie de le considérer comme un nombre (en cas de succès, il le dépose sur la pile, sinon il affiche UK ERROR).

On a donc mis dans le dictionnaire les nombres qui servent très souvent, pour qu'ils s'exécutent plus vite (en mode direct). Lorsqu'on tape 1, par exemple, c'est le mot du dictionnaire qui s'exécute, déposant le nombre 1 sur la pile.

De la même façon, la fréquence d'utilisation de certains mots justifie une définition optimisée. Par exemple, on utilise souvent l'opérateur « ajouter 1 », qui correspondrait en FORTH au mot :

```
: AJOUTER-1 1 + ; 33 AJOUTER-1 . 34  OK
```

(Cette opération porte même un nom propre, on parle d'incrémentation. C'est dire si elle sert souvent en informatique !!)
Il n'est pas illogique d'appeler ce mot 1+. Il s'agit d'une addition spécialisée, qui sait d'avance que c'est le nombre 1 qu'elle va ajouter. On comprendra facilement qu'on peut donc écrire un mot particulier, plus rapide d'exécution que le mot général (qui doit s'adapter à tous les cas possibles).

C'est ce qui a été fait, et nous disposons des mots suivants :

1+ n --- n+1 ajoute 1 au nombre du sommet de la pile.
1– n --- n–1 soustrait 1 au nombre du sommet de la pile.
2+ n --- n+2 ajoute 2 au nombre du sommet de la pile.
2– n --- n–2 soustrait 2 au nombre du sommet de la pile.
2* n --- 2×n multiplie le nombre du sommet de la pile par 2.
2/ n --- n/2 divise le nombre du sommet de la pile par 2.

Bien sûr, on peut utiliser 2 * au lieu de 2*, mais c'est moins rapide. Il est même plus rapide, pour multiplier par 64, de faire 6 fois 2* !!!
Par exemple :
La ligne `17 64 *` s'exécutera plus lentement que la ligne
`17 2* 2* 2* 2* 2* 2*` Incroyable, n'est-ce pas !

2-9 Quelques conseils

La plupart des débutants en FORTH vont certainement commencer à s'inquiéter en se disant : « Cette pile va être bien encombrée, comment vais-je faire pour m'y retrouver quand j'aurai un gros programme, avec beaucoup de calculs ? »

C'est vrai que si vous écrivez un « gros mot », vous risquez d'avoir des problèmes pour vous souvenir exactement de l'état de la pile à un moment donné, et surtout pour trouver les manipulations nécessaires pour l'amener dans l'état où vous voudriez qu'elle soit pour les opérations à effectuer.
Ça y est, vous vous êtes sortis de cette longue phrase ?... Et bien en FORTH, c'est pareil... Trois petites phrases sont souvent plus claires qu'une grande. Essayez de découper votre « programme » en **petits modules**, qui correspondront chacun à un mot. Ces mots n'utiliseront en général que deux ou trois valeurs sur la pile, et ne représenteront que quelques lignes de programme. **Documentez** bien vos mots (note ci-après), en indiquant en particulier les états de la pile (avant et après exécution), sur le papier ou en utilisant l'éditeur. Si une valeur est souvent utilisée, ne la laissez pas sur la pile ; définissez la comme **constante** ou comme **variable** (voir chapitre 3).

Documentation des programmes

Tout ce que vous taperez entre parenthèses sera complètement ignoré par FORTH. Mais bien sûr, la parenthèse ouvrante est un mot FORTH comme un autre, donc doit être suivie d'un espace.
: BEBEL BELL (BELL) BELL ; n'exécutera que deux bips...

C'est dire qu'entre ces parenthèses, on peut mettre tout ce qu'on veut, sans craindre les « UK ERROR ». En général, on y met des commentaires, mais d'autres utilisations sont imaginables.

La plupart des programmeurs en FORTH ont l'habitude d'indiquer systématiquement au moins les états de la pile.

Exemple :
(En utilisant pour PI la valeur approchée 22/7)

```
: PI* ( n1 --- n2 n2 = n1 × pi pi = 3,14...)
    22 * 7 / ;
: CIRC ( r --- c c = circonférence = 2 × pi × r)
    PI* 2* ;
: AIRE ( r --- s s = surface = pi × r × r)
    DUP * PI* ;
```

2-10 Le mot de la fin (du chapitre)

Le mot PI* que nous venons de définir contient un gros piège...

Exemple :

```
  100 PI* . 314  OK    Ok, rien à redire...
 1000 PI* . 3142 OK    Toujours d'accord...
                       (22/7 n'est qu'une valeur approchée)
10000 PI* . 3341 OK    Ah là, non !!! Pas OK du tout !!! (On devrait trouver 31428)
```

L'explication est simple, comme d'habitude. Notre mot PI* commence par multiplier par 22, puis divise par 7. Mais cela suppose que le produit du nombre du sommet de la pile par 22 soit un nombre entier entre −32768 et 32767. Le mot PI* ne donnera donc un résultat juste que pour les nombres inférieurs à 1489.
(En effet : 1489 × 22 = 32758 et 1490 × 22 = 32780)

Heureusement, (en FORTH, il y a toujours une solution), nous disposons d'un mot du dictionnaire qui, lui, ne tombe pas dans ce piège. Il s'agit du mot */ :
*/ (n1 n2 n3 --- n) calcule n1 × n2, et divise le résultat par n3

L'originalité de ce mot est que le résultat de la multiplication n'est pas un nombre comme ceux que nous avons étudiés jusqu'ici. Il s'agit d'une autre sorte de nombres que nous aborderons au chapitre 8, qui peuvent aller jusqu'à 2000000000... C'est dire que nous aurons de la marge !!!

Nous définirons donc PI* par :

```
: PI*  22  7  */  ;
```

Cette nouvelle définition de PI* nous donnera un résultat juste tant que le résultat final sera inférieur ou égal à 32767. Il faut pour cela :
n × 22 / 7 < 32768 donc n < 32768 * 7 /22

La valeur maximum de n donnant un résultat correct est 10425.

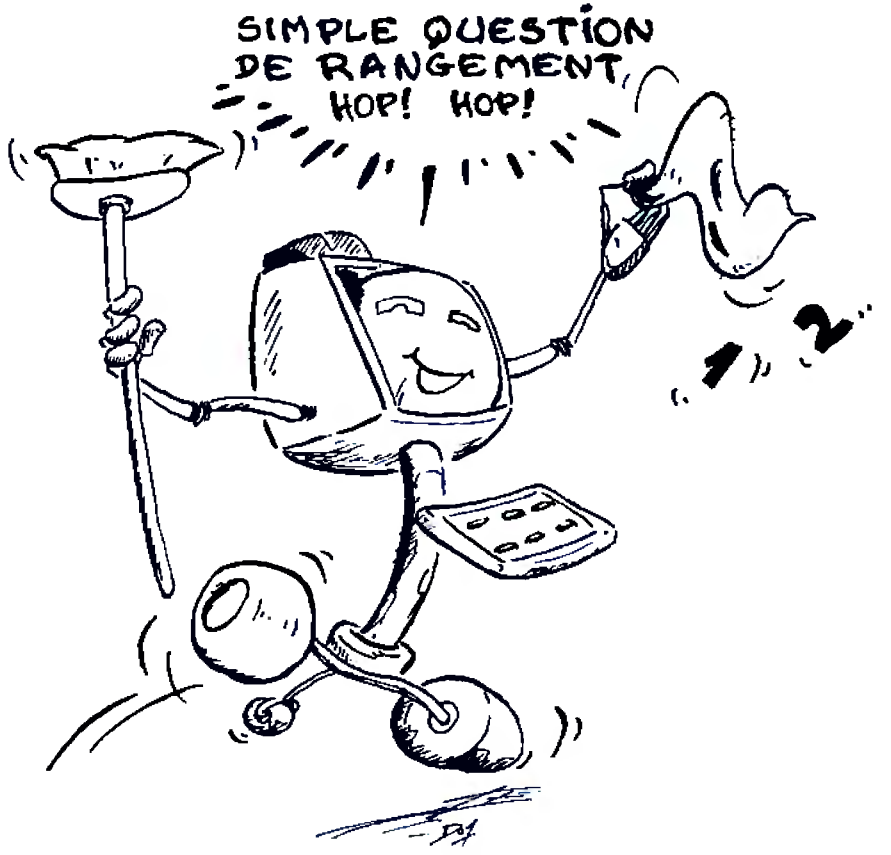
Exercices :

2.l Quelles valeurs du rayon r donnent des résultats cohérents, dans la version du 2-9 ?

2.m Mêmes questions, en remplaçant PI* par la version du 1-10 (en utilisant */)

Chapitre 3

Constantes et variables - tables et tableaux



La pile, c'est bien joli, mais ça n'est pas toujours indiqué pour ranger des quantités auxquelles on doit faire des références fréquentes.
Si nous empilons des nombres au fur et à mesure de notre application, cela nous obligera à tenir un compte précis de l'état de la pile pour en disposer le moment venu. Et les réarrangements de la pile seront pénibles et fastidieux, rendant nos mots difficiles à écrire et à corriger.

La solution est simple : il suffit de ranger ces nombres **ailleurs** que sur la pile, et les y ramener lorsqu'on en a besoin.
Pour cela, il faut pouvoir les nommer. Des objets qui portent un nom et qui ont une action (sur la pile...), ce sont des MOTS FORTH !!!

3-1 Constantes

Une constante est une quantité qui, en principe, ne change pas de valeur. Ce qu'on demande à une constante, c'est donc de déposer sa valeur sur la pile, afin que nous puissions l'utiliser pour nos calculs.

Définir une constante est simple. On précise sa valeur, on tape le mot **CONSTANT**, puis on lui attribue un nom.

Exemple :
25 CONSTANT LIGNES/ECRAN

C'est tout...
Pour utiliser cette constante, il suffit de taper son nom. La valeur 25 viendra automatiquement sur la pile.
Vérification :
LIGNES/ECRAN . 25 OK

Autre vérification :
VLIST... LIGNES/ECRAN est bien tout en haut du dictionnaire. Simple, non ? Et il faut bien reconnaître que lorsqu'on relit un mot, il est bien plus parlant de lire LIGNES/ECRAN que 25 !!!

Le dictionnaire de base contient bon nombre de constantes. Nous en rencontrerons au fur et à mesure de notre étude. Mais nous pouvons déjà donner quelques exemples :
Mieux !
RED est la constante 1 (couleur rouge)
Ø est la constante Ø (nombre Ø)
1, **2**, et **3** sont aussi des constantes.

Faisons un peu le point :
LIGNES/ECRAN est un mot du dictionnaire, dont l'action est de déposer la valeur 25 sur la pile. Ce mot a été fabriqué en utilisant le mot CONSTANT. CONSTANT est donc un mot d'un type spécial, qui sert à fabriquer d'autres mots, tout comme le mot :.

Les mots de ce type s'appellent des **mots de définition** (ou encore mots de compilation). Un des points forts du FORTH est de pouvoir définir de tels mots, qui eux-mêmes définissent des mots.
La question est un peu délicate pour un manuel d'initiation, mais vous pourrez en avoir une idée en lisant l'annexe C.

3-2 Variables

Ici, le problème est un peu différent. Les variables sont des quantités qui, en principe, changent souvent de valeur. On a donc besoin non seulement de lire cette valeur, mais aussi de la modifier.

Pour cela, on utilise non pas la valeur elle-même, mais son **adresse**, (nombre rappelant l'endroit où elle est entreposée).

Ce concept d'adresse est fondamental en FORTH, au même titre que les concepts de pile et de dictionnaire. En fait, il est fondamental dans n'importe quel langage, puisque c'est lui qui est à la base du fonctionnement des ordinateurs (on parle aussi parfois de « pointeurs »).
En FORTH, contrairement à d'autres langages, c'est à vous de manipuler les adresses.
On définit des variables en utilisant le mot **VARIABLE.**

Exemple :
Mettons-nous dans la peau d'un épicier qui vend des boîtes de conserve, entre autres des boîtes de choucroute.

```
VARIABLE RESERVE-CHOUCROUTE ( Nombre de boîtes à la réserve)
VARIABLE RAYON-CHOUCROUTE ( Nombre de boîtes en rayon)
```

Essayons...

```
RESERVE-CHOUCROUTE . 25298  OK
```

Ce nombre, 25298 (vous n'avez peut-être pas le même, cela n'a pas d'importance), n'est pas la valeur du stock, mais l'*adresse* où est rangée cette valeur. On voit tout de suite que les mots les plus utilisés du FORTH seront certainement les mots « lire-à-une-adresse » et « écrire-à-une-adresse ». Ils font l'objet de notre prochain paragraphe.

En attendant, vérification de routine :
VLIST... RESERVE-CHOUCROUTE et RAYON-CHOUCROUTE sont bien en haut du dictionnaire.

Notre dictionnaire contient aussi déjà des variables en bon nombre. Citons par exemple :

WIDTH longueur maximum autorisée des mots du dictionnaire.
SCR numéro du dernier écran avec lequel a travaillé l'éditeur.
C/L nombre de caractères par ligne (par exemple 80 pour l'imprimante).
L/SCR nombre de lignes par écran (ou par page, pour l'imprimante).

Et bien d'autres que nous verrons petit à petit.

VARIABLE, comme CONSTANT ou :, sert à définir d'autres mots. C'est donc un mot de définition.

3-3 Manipuler des variables

Il s'agit ici de lire ou d'écrire un nombre dans une variable. Comme ces opérations sont très fréquentes, on leur a choisi des noms courts, au détriment de la lisibilité des applications.
S'ils vous déplaisent, vous pouvez toujours les nommer autrement. Ces noms sont toutefois standards, et il serait bon que vous vous y habituiez.

3-3-1 Lire le contenu d'une variable

L'action de déposer sur la pile la valeur d'une variable est réalisée par le mot @. Comment prononcer ce mot ?!!... Les Anglais disent « fetch », alors pourquoi pas « aller chercher » ?
Son fonctionnement est le suivant :
On tape le nom de la variable (puis un blanc, bien entendu), suivi du mot @.

Exemple :
```
WIDTH @ . 31  OK
L/SCR @ . 25  OK
C/L @ . 40  OK
RESERVE-CHOUCROUTE @ . Ø  OK
RAYON-CHOUCROUTE @ . Ø  OK
```

Ça n'est vraiment pas compliqué, non ?!

En général, on ne demande pas l'affichage du contenu de la variable, on se contente d'aller chercher sa valeur et de l'amener sur la pile (par @) pour l'utiliser. Toutefois FORTH est largement interactif, et on travaille souvent en mode direct. La séquence @ . est souvent utilisée au clavier, pour vérifier le contenu d'une variable.

Taper 2 touches étant deux fois plus long que taper une seule touche, (même en FORTH), on a donc défini un mot, qui pourrait s'appeler « Interroger une variable », et qui s'écrit ? (point d'interrogation).
```
RESERVE-CHOUCROUTE ? Ø  OK
WIDTH ? 31  OK
```

Au fait; avez-vous remarqué que le contenu des variables que nous venons de définir (RESERVE-CHOUCROUTE et RAYON-CHOUCROUTE) était Ø ???
Eh bien c'est ainsi, lorsqu'on crée une variable par le mot VARIABLE, elle contiendra toujours la valeur Ø. On dit qu'elle est automatiquement « initialisée » à Ø.
Il ne faudra donc pas oublier, dans vos applications, de lui donner la valeur initiale que vous souhaitez, si cette valeur n'est pas Ø.

3-3-2 Modifier le contenu d'une variable

Pour mettre un nombre dans une variable (ou initialiser une variable à une valeur donnée), c'est tout aussi simple :
On tape le nombre, un espace, le nom de la variable, un espace, et la commande FORTH d'affectation est ! (point d'exclamation).

Exemple :
```
23 RESERVE-CHOUCROUTE !  OK
   (23 boîtes de choucroute à la réserve)
```

Vérification :
RESERVE-CHOUCROUTE ? 23 OK
7 RAYON-CHOUCROUTE ! RAYON-CHOUCROUTE ? 7 OK

Quoi de plus simple ?

Autre exemple : Voulez-vous que votre VLIST n'affiche qu'un nom par ligne ? C'est facile, car le mot teste après chaque affichage la position sur la ligne. Définissons donc :
: VLIST+ 2 C/L ! VLIST 40 C/L ! ; VLIST+ ...

Dans le fond, ces noms ne sont pas si terribles à retenir... On peut par exemple se souvenir que :
? pour interroger.
! pour affirmer... (JE VEUX la valeur xxx dans la variable vvv)

Quelles sont les actions sur la pile des mots @ et ! ???
Nous savons déjà qu'une variable laisse son adresse sur la pile. Par conséquent, ils utilisent la pile de la manière suivante :
@ adr --- n (n = nombre situé à l'adresse adr)
! n adr --- (range n à l'adresse adr)

Pourquoi avoir écrit adr comme contenu de la pile ? Une adresse est un nombre, mais on en fait un usage particulier. Ecrire adr facilite la relecture des commentaires. On pourrait aussi bien écrire :
! n1 n2 --- mais ce serait beaucoup moins parlant...

Dernière remarque : Attention à l'ordre, lorsque vous utilisez le mot !. Il range n à l'adresse adr, et non adr à l'adresse n (??!!!...). C'est une erreur courante lorsqu'on débute, par toujours facile à remarquer, et qui peut donner des résultats extrêmement curieux...

3-3-3 Utiliser des variables

Usuellement, on amène sa valeur sur la pile pour la modifier, puis on réécrit la nouvelle valeur dans la variable.

Exemples :
Pour faire l'inventaire des boîtes de choucroute :
RESERVE-CHOUCROUTE @
RAYON-CHOUCROUTE @ + . 30 OK

Transporter 12 boîtes de la réserve au magasin :
RESERVE-CHOUCROUTE @ 12 - RESERVE-CHOUCROUTE !
RAYON-CHOUCROUTE @ 12 + RAYON-CHOUCROUTE !

Ou encore, on définira un mot :
: RESERVE->RAYON-CHOUCROUTE (n ---)
RESERVE-CHOUCROUTE @ OVER - RESERVE-CHOUCROUTE !
RAYON-CHOUCROUTE @ SWAP + RAYON-CHOUCROUTE ! ;

Et à la caisse, on utilisera un mot qui tienne à jour l'état du rayon. Par exemple, le mot CHOUCROUTE :

```
: CHOUCROUTE (n ---)
RAYON-CHOUCROUTE @ SWAP - RAYON-CHOUCROUTE ! ;
```

Notre commerçant n'a plus maintenant qu'à taper 5 CHOUCROUTE lorsqu'un client vient lui acheter 5 boîtes, pour qu'aussitôt le nombre de boîtes restant en rayon soit diminué de 5 (dans l'ordinateur...).

Cette opération « ajouter (ou retrancher) un nombre au contenu d'une variable » intervient si souvent qu'on a créé un mot spécialisé, qui s'appelle +! .

+! n adr --- additionne n au nombre qui se trouve à l'adresse adr

On pourrait réécrire ainsi les mots précédents :

```
: RESERVE->RAYON-CHOUCROUTE ( n ---)
            DUP RAYON-CHOUCROUTE +!
            NEGATE RESERVE-CHOUCROUTE +! ;
```

Et :

```
CHOUCROUTE              ( n ---)
            NEGATE RAYON-CHOUCROUTE +! ;
```

De même, pour ramener les boîtes du rayon à la réserve (réfection du rayon) :

```
: RAYON->RESERVE-CHOUCROUTE ( ---)
            RAYON-CHOUCROUTE @ RESERVE-CHOUCROUTE +! ;
```

3-3-4 Un exemple pris dans FORTH : La variable BASE

FORTH ne travaille (pour le moment) qu'avec des nombres entiers entre −32768 et 32767, c'est entendu... Mais il peut lire et afficher ces nombres dans n'importe quelle base !!! Vous souvenez-vous de ce qu'est une base de numération ?

Très succinctement, nous comptons d'habitude en base dix (comme les doigts de la main), en utilisant pour écrire les nombres les dix symboles Ø, 1, 2, 3, 4, 5, 6, 7, 8 et 9.

Si nous n'avions qu'une main, nous pourrions compter en base cinq, en utilisant les symboles Ø, 1, 2, 3, 4. Pas de changement pour les cinq premiers nombres, le nombre « cinq » s'écrira alors 1Ø (une main). Le nombre « six » s'écrirait 11 (une main et un doigt). Le nombre que nous avons l'habitude de prononcer « vingt-trois » correspondrait à quatre mains et trois doigts, et s'écrirait donc 43.

Comment écrire le nombre « vingt-cinq » ? Eh bien, il correspond à « cinq mains », c'est-à-dire à « une main de mains » (si vous préférez, une poignée de mains). Il est impossible de l'écrire avec seulement 2 chiffres, nous l'écrirons 1ØØ. Le « 1 » signifie 1 unité de la troisième colonne. Les unités de la deuxième colonne sont les « mains », celles de la troisième colonne des « poignées de main », celles de la quatrième colonne seraient des « poignées de poignées de mains » (et correspondraient donc à cinq fois cinq fois cinq doigts, c'est-à-dire à cent vingt-cinq doigts, comme nous disons d'habitude).

Et ainsi de suite...
On peut aussi travailler en base deux (coupez-vous trois doigts... La suite des nombres sera alors Ø, 1, 1Ø, 11, 1ØØ, 1Ø1, 11Ø, 111, 1ØØØ, ...), ou dans n'importe quelle autre base, le tout est d'avoir le bon nombre de doigts. Les ordinateurs utilisent tous la base deux.

Lorsque FORTH affiche un nombre ou lit un nombre, il consulte la variable BASE, qui contient la base en cours d'utilisation (pour lecture-affichage).

Exemples :

```
BASE ? 1Ø  OK
2 BASE ! 5 .
5 ? UK ERROR   (5 n'est pas un chiffre en base deux)
BASE ? 1Ø  OK                (Attention !!! voir remarque ci-après...)
                             (La base est bien deux, mais en base deux, le nombre
                             deux s'écrit 1Ø)
DECIMAL                      (Ce mot remet FORTH en base dix...)
5 2 BASE ! . 1Ø1  OK         (Ah, quand même !!!!)
DECIMAL
```

Voulez-vous des affichages en base trois, sans modifier la base avec laquelle vous êtes en train de travailler ? (Bref, un mot de « conversion en base trois »).
Fabriquons le mot suivant :

```
: .(BASE-3) ( n ---)
        BASE @ ( pour se souvenir de la base)
        3 BASE ! ( passe en base 3)
        SWAP .  ( affiche n en base 3)
        BASE ! ; ( remodifie la base)
```

Puis

```
19 .(BASE-3) 2Ø1  OK
```

Les bases qui servent le plus souvent en informatique sont la base dix et la base seize (système hexadécimal, les symboles sont Ø, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F).
Pour simplifier la tâche, nous disposons donc de 2 mots du dictionnaire :
DECIMAL (---) Pour travailler en système décimal.
HEX (---) Pour travailler en système hexadécimal.

Les symboles utilisés : Ce sont dans l'ordre les chiffres de Ø à 9, puis, le cas échéant, les lettres de l'alphabet (majuscules, puis minuscules,...). Par exemple, en base vingt (vingt symboles), les symboles utilisés seront Ø, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, G, H, I, J.

Remarque importante :
Si vous êtes en base 5, par exemple, et que vous tapez la commande 1Ø BASE ! vous serez encore en base 5 !!!
Pourquoi ? Eh bien tout simplement parce que le « 1Ø » que vous avez tapé a été lu en utilisant la base cinq !!! Pour vous mettre en base dix, il faut taper : 2Ø BASE ! ... Vous voyez tout de suite l'utilité des mots DECIMAL et HEX, qui, eux, ne soulèveront pas ce problème !!!

Reconnaissez que si vous êtes en base 2, il est plus simple de taper DECIMAL que la commande « 1Ø1Ø BASE ! »...

Exercices :

3.a - Comment s'écrivent les nombres suivants en base 2 ? en base 16 ? (Nombres donnés en décimal) 2334, 4567, 34, −546, 7680, 32767.
3.b - Inversement, un nombre s'écrit 7FFF en hexadécimal. Comment s'écrit-il dans le système décimal ?

3-4 Tableaux

Reprenons l'exemple précédent. Notre épicier vend aussi des boîtes de cassoulet, de couscous, de potée, et de raviolis...
Nous pourrions, comme ci-dessus, tenir un état des quantités disponibles en réserve pour chacun de ces articles, en utilisant des variables RESERVE-CASSOULET, RESERVE-COUSCOUS, RESERVE-POTEE, et RESERVE-RAVIOLIS.

Mais cela va faire grossir inconsidérément le dictionnaire, et surtout nous faire écrire 5 fois la même opération...
L'idée est de présenter les choses à l'ordinateur exactement comme on le ferait sur une feuille de papier, dans un tableau :

RESERVE-CONSERVES
1 — CHOUCROUTE 23 boîtes
2 — CASSOULET 35 boîtes
3 — COUSCOUS 15 boîtes
4 — POTEE 30 boîtes
5 — RAVIOLIS 37 boîtes

Nous voulons donc définir une entité RESERVE-CONSERVES, qui puisse contenir 5 nombres, et les utiliser comme des variables (chacun de ces nombres ayant une signification précise, comme ci-dessus).
Cela se fait par le mot **ARRAY** (qui est encore un mot de définition, puisqu'il permet de définir d'autres mots).
5 ARRAY RESERVE-CONSERVES crée un tableau pouvant contenir 5 nombres.

Pour remplir ce tableau :
13 comme premier nombre (position Ø)
13 Ø RESERVE-CONSERVES ! Range 13 en position Ø dans RESERVE-CONSERVES

De même :
35 1 RESERVE-CONSERVES !
15 2 RESERVE-CONSERVES !
30 3 RESERVE-CONSERVES !
37 4 RESERVE-CONSERVES !

Mais, direz-vous, comment savoir quel nombre correspond aux boîtes de choucroute, et quel nombre aux boîtes de cassoulet ?...
Lors de l'écriture de votre application, il faudra effectivement que vous vous souveniez que les boîtes de choucroute sont l'article numéro Ø dans le tableau, les boîtes de cassoulet le numéro 1, etc.
Si cela vous gêne, vous pouvez toujours définir ces mots comme étant des constantes :

```
Ø CONSTANT BOITES-CHOUCROUTE
1 CONSTANT BOITES-CASSOULET
```

etc.

Puis, par exemple :
Le grossiste vient de livrer 11 boîtes de cassoulet. Elles sont à la réserve.

```
11 BOITES-CASSOULET RESERVE-CONSERVES +!
```

Combien y a-t-il de boîtes de cassoulet à la réserve ?

```
BOITES-CASSOULET RESERVE-CONSERVES ?
```

Mais cela n'est peut-être pas indispensable... Au moment où vous écrivez votre application, vous connaissez ces positions, il suffit, lorsque vous parlerez de boîtes de choucroute, de remplacer partout « RESERVE-CHOUCROUTE » par « Ø RESERVE-CONSERVES ». De la même façon, pour parler de boîtes de cassoulet, au lieu d'utiliser « RESERVE-CASSOULET », on utilisera « 1 RESERVE-CONSERVES ». Ce nombre, Ø, 1, 2, 3, ou 4, peut très bien être laissé sur la pile par le mot utilisateur (par exemple CHOUCROUTE), de sorte que nous n'aurons plus qu'un seul mot de mise à jour.

Passons à la pratique. Nous allons créer des mots CHOUCROUTE+, CASSOULET+, COUSCOUS+, POTEE+, et RAVIOLIS+ (+ pour « Livraison »), qui mettent automatiquement à jour le nombre de boîtes de conserves de la réserve lors du passage du livreur.
On tapera par exemple :

```
10 CASSOULET+
```

pour dire « On vient de me livrer 10 boîtes de cassoulet. Je les ai mises à la réserve », et le stock sera immédiatement mis à jour (augmenté de 10)

Réalisation

```
: LIVRAISON             ( n p -- p = position à modifier dans le tableau)
        RESERVE-CONSERVES ( p RESERVE-CONSERVES     adresse du nombre à modifier)
        +!                (effectue la modification dans le tableau)
: CHOUCROUTE+           ( n --- ) Ø LIVRAISON ;
: CASSOULET+            ( n --- ) 1 LIVRAISON ;
: COUSCOUS+             ( n --- ) 2 LIVRAISON ;
: POTEE+                ( n --- ) 3 LIVRAISON ;
: RAVIOLIS+             ( n --- ) 4 LIVRAISON ;
```

Bien que le mot LIVRAISON ne soit pas indispensable, reconnaissez qu'il facilite nettement la relecture de cette petite application. Cela est aussi dans la « philosophie » du FORTH.

Il suffit maintenant à notre commerçant de taper lors de la livraison :
5 CHOUCROUTE+ 8 COUSCOUS+ 8 CASSOULET+ 3 POTEE+
pour qu'aussitôt le stock soit mis à jour.

Vérification :
0 RESERVE-CONSERVES ? 28 OK (23 + 5 = 28)
1 RESERVE-CONSERVES ? 43 OK (35 + 8 = 43)
etc.

Exercices :

3.c - Il serait intéressant de regrouper ces interrogations dans un même mot, par exemple le mot ETAT-CONSERVE, ou RESERVE-CONSERVES-?, qui puisse afficher :
Choucroute : 28
Cassoulet : 43
Couscous : 23
Potée : 33
Raviolis : 37
Nous vous laissons écrire ce mot à titre d'exercice. Notre vendeur peut donc maintenant taper ETAT-RESERVE pour savoir combien il doit commander de boîtes...
Quel langage peut en faire autant, de manière si concise et interactive ?

3.d - Faire un tableau RAYON-CONSERVES donnant les boîtes en rayon, et des mots CASSOULET–, COUSCOUS–, etc. Il faut qu'à la caisse on puisse taper : 2 POTEE– 1 COUSCOUS– 3 RAVIOLIS–
lorsque Mme Dupont viendra faire ses courses. Le tableau RAYON-CONSERVES devra alors être modifié en conséquence.

3.e - Les mots CHOUCROUTE, CASSOULET, etc. étant définis comme constantes (0, 1, etc.), faire des mots R–>M et M–>R qui permettent de transférer des boîtes de la réserve au magasin (R–>M) et vice versa.
On devra pouvoir taper :
5 CHOUCROUTE R–>M 6 CASSOULET R–>M etc.
et les deux tableaux RESERVE-CONSERVES et RAYON-CONSERVES devront être mis à jour.

3-5 Tables

Les tables sont aux tableaux ce que les constantes sont aux variables. On range dans les tables des quantités qui en principe ne changent jamais de valeur. Définissons par exemple une table pour les prix des boîtes de conserve de notre magasin.
Cela se fera tout simplement en disant :
TABLE PRIX-CONSERVES

Pour remplir cette table, on utilise le mot « , » (virgule), qui met un nombre dans le dictionnaire à la position courante (Cette question de « position courante » est un peu plus compliquée, et ne sera pas abordée ici). Quoi qu'il en soit, notre problème est de mettre des nombres dans le dictionnaire, et « , » les y mettra à la bonne place, à condition de le faire maintenant (avant d'avoir défini un autre mot).
La suite sera donc par exemple :

```
14 , (prix d'une boîte de choucroute)
15 , (prix d'une boîte de cassoulet)
17 , (prix d'une boîte de couscous)
11 , (prix d'une boîte de potée)
10 , (prix d'une boîte de raviolis)
```

Puis :

```
3 PRIX-CONSERVES . 11  OK
```

C'est bien le prix d'une boîte de potée, qui est la rubrique numéro 3 (la quatrième rubrique) de la table.

Remarques :

— L'utilisation des tables se fait comme celles des tableaux, à cela près que la valeur laissée sur la pile est bien la valeur d'un élément de la table (comme pour une constante) et non son adresse (comme pour une variable, ou comme le ferait un tableau).

— Attention, comme pour les tableaux, le premier article porte le numéro Ø. Vous pouvez ne pas utiliser l'article Ø si vous voulez ne pas vous tromper, mais il ne faudra pas oublier de réserver la place correspondante, en y mettant une valeur quelconque, par exemple le nombre d'articles différents...
Ce qui conduit à :

```
   TABLE PRIX-CONSERVES
5 , (5 rubriques) 14 , 15 , 17 , 11 , 10 ,
```

Il serait raisonnable, dans ce cas, de procéder de la même façon en ce qui concerne le tableau RESERVE-CONSERVES, et aussi le tableau RAYON-CONSERVES...

Exercice :

3.f - Vous avez maintenant tout ce qu'il faut pour compléter l'exemple précédent, de manière à être utilisable à la caisse... Nous vous suggérons de définir une variable « TOTAL-A-PAYER », et de modifier les mots précédents pour que le commerçant n'ait à taper que :

```
VENDU (Initialisation de TOTAL-A-PAYER)
2 CHOUCROUTE 1 CASSOULET 1 RAVIOLIS 1 POTEE
```

pour que soit mis à jour le nombre de boîtes restant en rayon, et pour qu'apparaisse sur son écran le détail des achats, leur montant, et le total. Nous verrons au prochain chapitre comment modifier ces mots pour que l'ordinateur prévienne si le rayon est vide, et s'il faut commander des boîtes au grossiste...

Si vous avez une imprimante, vous pouvez même vous amuser à imprimer le ticket de caisse, il suffit d'insérer dans vos mots le mot PRINTER lorsque vous voudrez que les affichages ultérieurs (texte, nombre, mais aussi passage à la ligne par CR) soient imprimés, et d'insérer le mot CONSOLE pour que les affichages se passent à l'écran. Ceci sera revu ultérieurement. Toujours dans les améliorations possibles, on peut aussi définir des « ardoises » pour les clients. Il suffirait alors de faire un mot EN-COMPTE, qui permettrait de taper : « Dupont EN-COMPTE » pour que le compte de Mme Dupont soit augmenté du montant à payer (et bien sûr des mots comme COMPTE? pour consulter les ardoises des clients, et REGLEMENT-COMPTE pour le jour où Mme Dupont vient régler ses dettes).

Chapitre 4

Conditions et répétitions

Il s'agit ici de mots permettant d'agencer les autres mots entre eux, et de modifier le déroulement habituel des opérations (qui traite les mots « séquentiellement », dans l'ordre où ils ont été écrits).
Pour cette raison, les mots étudiés dans ce chapitre sont appelés en général « mots de contrôle » (sous-entendu « contrôle du déroulement des opérations »)

Ce chapitre se divise en deux parties bien distinctes :
— Décider du choix d'une action ou d'une autre...
— Répéter une action un certain nombre de fois...

4-1 Expression des conditions

Le principe de la décision est le suivant : si une certaine condition est réalisée, on fait une action, sinon on en fait une autre (ou rien du tout).
La question qui se pose est tout d'abord : comment l'ordinateur saura-t-il si la condition est réalisée ou non ?

Voyons par exemple le mot 0=, qui répond à la question :
« Le nombre qui est sur la pile est-il nul ? »

```
23 0=  OK Tiens, où est la réponse ?
. 0  OK Sur la pile, bien entendu !!!
```

Essayons autre chose :

```
45 0= . 0  OK
1  0= . 0  OK Mais il répond toujours 0 ???!!!
0  0= . 1  OK Non !!!
```

Moralité :
0 signifie « FAUX, cela ne vaut rien »
1 signifie « VRAI »

En fait, tout nombre non sul signifie pour FORTH : « VRAI, condition RÉALISÉE ».

Les mots de base qui serviront à écrire les conditions sont :

```
0=  ( n --- cond)  cond vaut 1 si n=0, 0 sinon.
0<  ( n --- cond)   "    "  1 si n<0, 0 sinon.
0>  ( n --- cond)   "    "  1 si n>0, 0 sinon.
=   ( n p --- cond)  "    "  1 si n=p, 0 sinon.
<   ( n p --- cond)  "    "  1 si n<p, 0 sinon.
>   ( n p --- cond)  "    "  1 si n>p, 0 sinon.
```

Essayez-les pour bien comprendre, par exemple :

```
3 5 = .0  OK
-3 -4 > .1  OK (Ça va, pas trop surpris ?)
```

Si vous n'êtes pas familier avec cette méthode d'expression des conditions (par des 1 ou des 0), n'hésitez pas à faire de nombreux essais. Votre ordinateur ne le répétera à personne...
Nous reviendrons au 4-4 sur le calcul des conditions, mais pour le moment restons-en là.

4-2 Choisir son action

Le schéma est le suivant :

```
..... (actions antérieures)
La condition est-elle vraie ? (à vous de mettre ce qu'il faut sur la pile)
IF                  (Si oui)
..... (actions à exécuter si la condition est vraie)
ELSE                (Si non)
..... (actions à exécuter si la conditions est fausse)
THEN                (fin des actions soumises à la condition)
..... (suite du mot...)
```

Attention : IF, THEN et ELSE ne sont utilisables que dans la définition d'un mot.

Par rapport à d'autres langages, cette construction (THEN à la fin) est certainement un peu déroutante (d'ailleurs, les premiers FORTH utilisaient ENDIF au lieu de THEN), mais elle est conforme à la logique du langage.

Il arrive fréquemment que lorsque la condition est fausse, on n'ait aucune action à exécuter. Dans ce cas, on peut omettre le ELSE, et on obtient le schéma suivant :

```
..... (actions antérieures)
La condition est-elle réalisée ?
IF                  (si oui)
..... (actions à exécuter si la condition est réalisée)
THEN                (fin des actions soumises à la condition)
..... (suite de la définition en cours)
```

Exemples :
Pour vous aider à vérifier vos conditions (nous réutiliserons TEST au 4-4)
Imaginons que n soit le résultat de l'évaluation d'une condition...

```
: TEST (n ---)
        IF
        ." VRAI"
        ELSE
        ." FAUX"
        THEN CR ;
```

Testons ce mot pour vérifier notre compréhension :

```
0 TEST FAUX                1 TEST VRAI                22 TEST VRAI
OK                         OK                         OK
```

Autre point bien agréable du FORTH : vous pouvez tester vos mots un à la fois, au fur et à mesure de leur écriture. C'est très commode pour rechercher les erreurs lorsqu'une de vos applications ne fonctionne pas comme prévu.

Pour voir si un nombre est divisible par 7 :

```
: DIV/7? (n ---)
        DUP                          (duplique pour pouvoir afficher n)
        7 MOD                        (reste de la division de n par 7)
        0=                           (le reste est-il nul ?)
        IF CR . ." est divisible par 7"
        ELSE CR . ." n'est pas divisible par 7"
        THEN ;
```

Ou encore, puisque tout nombre non nul est considéré comme condition vraie :

```
: DIV/7?? (n ---)
        DUP 7 MOD          (« condition vraie ? » signifie « reste non nul ? »)
        CR CR .            (Affichage à faire dans les deux cas)
        IF ." n'est pas"
        ELSE ." est"
        THEN
        ." divisible par 7" ;
```

Voulez-vous faire une division ? Oui, mais n'oubliez pas que FORTH ne fera aucune vérification... Il est donc prudent d'écrire par exemple :

```
: //        (n1 n2 --- n3 division à toute épreuve)
    DUP 0=
    IF      ." Je ne veux pas diviser par 0"
            DROP                          (pour enlever n2 de la pile)
    ELSE
    THEN ;
```

Ici encore, DUP est nécessaire car "0= IF" détruit le diviseur. DROP enlève ce diviseur si on ne fait pas la division. De sorte que dans tous les cas, le mot // laissera sur la pile **un seul** nombre, qui sera le quotient si le diviseur n'est pas nul, et le dividende dans le cas où le diviseur est nul (pourquoi pas... On pourrait aussi choisir de laisser dans ce cas le nombre − 1 sur la pile, par exem-

ple. Il suffirait pour cela de remplacer « DROP » par « DDROP −1 »).
Ou encore, bien sûr :

```
: // DUP IF / ELSE ." Non, je ne diviserai pas par zéro" DROP THEN ;
```

Un dernier exemple sans ELSE, toujours à propos de division :
Nous voulons que l'ordinateur nous prévienne lorsqu'il a divisé par Ø, mais qu'il continue son travail quand même :

```
: ///          ( n1 n2 --- n3 division avec mise en garde)
      DUP      ( on aura encore besoin du diviseur)
      Ø=
      IF       BELL CR CR ." Attention, j'ai divisé par zéro"
      THEN
      / ;
```

Vous pouvez utiliser ces divisions dans vos autres mots, pour travailler avec plus de sécurité. Dans le cas d'une division par Ø, vous pouvez au choix afficher un message, arrêter l'exécution, ou même exécuter des mots de votre choix. FORTH vous laisse choisir votre traitement des erreurs...

En résumé :
— **La condition s'écrit avant le IF**, conformément à la logique du FORTH.
— **IF-ELSE-THEN, ou IF-THEN, forment un tout.** IF n'a pas de sens s'il n'est pas suivi de THEN. ELSE ne peut être qu'entre un IF et un THEN. Quant à THEN, il sert aussi à marquer la fin d'une autre structure de sélection (voir Manuel de Référence). Lors de la mise au dictionnaire d'une définition incorrecte, FORTH vous répondra :
? UF ERROR (Unfinished) « Il y a bien un IF, mais je n'ai pas trouvé de THEN »
? CD ERROR (Conditional ?) « J'ai un ELSE ou un THEN, mais où donc est le IF ?? »
— Les mots IF, THEN et ELSE sont des mots d'un genre nouveau. Ils s'exécutent lors de la mise au dictionnaire d'un mot, et non lors de son exécution. Un tel mot s'appelle un mot *immédiat* (Cette question sera abordée à l'annexe C). Cela a pour principale conséquence qu'**on ne peut utiliser IF, THEN ou ELSE que dans une définition**, entre : et ;. Il est impossible de les utiliser en mode interactif. FORTH nous répondrait « ? CO ERROR » (Compilation only) (mode compilation uniquement) ou « ? CD ERROR ? » (pour ELSE).

4-3 Avec des si...

Les actions soumises à des conditions peuvent aussi être des actions conditionnelles.
On aura alors des « IF », des « ELSE », et des « THEN » imbriqués les uns dans les autres.
Reprenons l'exemple du § 3-3-3 (variables RAYON-CHOUCROUTE et RESERVE-CHOUCROUTE).
Nous voulons qu'un client puisse faire une commande téléphonique à la caisse, pour une livraison ultérieure. L'ordinateur doit nous répondre si nous pouvons satisfaire cette

commande, et le cas échéant s'il faut retirer des boîtes du rayon (pour une livraison, priorité à la réserve).

```
: COMMANDE-CHOUCROUTE ( n --- n ; n = nombre de boîtes commandées)
    DUP RESERVE-CHOUCROUTE @          ( Nombre de boîtes de la réserve)
    >                                 ( La réserve est-elle insuffisante ?)
    IF RESERVE-CHOUCROUTE @ -         ( Si oui, la différence est-elle)
       DUP RAYON-CHOUCROUTE @         ( supérieure au nombre de boîtes)
       >                              ( actuellement en rayon ?)
       IF ." Impossible" DROP         ( Oui, la livraison est impossible.)
       ELSE                           ( Non... C'est possible, mais il faudra)
            INVCOLOR ." Enlever immédiatement"   ( mettre tout de suite)
            . ." boîtes du rayon"      ( de côté x=commande-réserve)
            INVCOLOR BELL BELL BELL    ( boîtes, hors du rayon)
       THEN
ELSE ." Commande possible"             ( Réserve suffisante, pas de problème.)
THEN ;
```

Notez la présentation, avec des **indentations** (décalages dans l'écriture). Il est en effet essentiel, pour que FORTH s'y retrouve, que si on emboîte des expressions conditionnelles, celles-ci ne se chevauchent pas. Cela est mis en évidence par l'indentation ; on voit très bien à quels « IF » sont associés chacun des « THEN » ou « ELSE ».

On aurait pu écrire le mot sous la forme suivante :

```
: COMMANDE-CHOUCROUTE ( n --- n ; n = nombre de boîtes commandées)
  RESERVE-CHOUCROUTE @ - DUP 0> IF DUP RAYON-CHOUCROUTE @ >
IF ." Impossible" DROP ELSE BELL BELL INVCOLOR ." Enlever" . ." Boîtes
du rayon" INVCOLOR THEN ELSE ." Commande possible" DROP THEN ;
```

FORTH n'en aurait pas été gêné, pour lui cela n'aurait rien changé. Mais pour nous, s'il y avait une erreur (de frappe, bien entendu), avouez qu'elle serait plus difficile à déceler !!!

Il y a une règle générale pour les imbrications de IF...THEN...ELSE : Chaque ELSE ou THEN est associé au dernier IF qui n'a pas encore été annulé par un THEN. Il est donc incorrect, par exemple, d'avoir dans un mot deux ELSE sans un THEN (ou un IF) entre les deux (Cela ferait deux ELSE pour un même IF).

Vous pouvez réfléchir à cette règle, mais l'utilisation correcte d'indentations la rend tout à fait inutile...

Par ailleurs, vous courrez moins de risques en écrivant des mots plus courts et plus simples. L'exemple précédent pourrait être traité ainsi :

```
: PREPARER-BOITES (x --- ; x = nombre de boîtes à mettre de côté)
      INVCOLOR ." Mettre" . . " boîtes hors du rayon"
      INVCOLOR BELL BELL BELL ;
```

Il sera donc intéressant d'avoir x sur la pile. Noter que x = commande-réserve.

```
: RESERVE-INSUFFISANTE ( x --- ; puisque x doit servir... )
        DUP RAYON-CHOUCROUTE @ >    ( Commande - Réserve > Rayon ?)
        IF                          ( Si oui, Commande > Stock total)
                DROP ." Commande impossible"    ( DROP pour supprimer x)
        ELSE PREPARER-BOITES
        THEN ;
```

Et enfin :

```
: COMMANDE-CHOUCROUTE ( n ---)
        RESERVE-CHOUCROUTE @ -          ( x est déjà sur la pile !)
        DUP 0>                          ( Commande > Réserve ?)
        IF RESERVE-INSUFFISANTE
        ELSE ." Commande possible"      ( ou TRAITER-COMMANDE...)
        THEN ;
```

Peut-être un peu plus long, mais combien plus lisible et expressif !!! Il y a, de manière générale, un grand intérêt à nommer les actions intermédiaires dès qu'elles correspondent à des situations concrètes. C'est une des grandes qualités du FORTH que de permettre cela.

Exercice :

4.a - Réécrire l'application précédente de manière qu'un client puisse commander n'importe quelle sorte de conserve (utiliser les tableaux RESERVE-CONSERVES et RAYON-CONSERVES).

4.b - On demande en outre que l'ordinateur annonce le montant de la commande (utiliser la table PRIX-CONSERVES).

4.c - Faire un mot ORDRE (a b c --) qui affiche les nombres a, b, c dans l'ordre croissant (du plus petit au plus grand).
On devra pouvoir taper par exemple : 65 12 23 ORDRE 12 23 65 OK

4-4 Combiner des conditions

Nous nous intéressons ici aux conditions multiples, du genre :
Est-ce que les conditions A et B sont toutes les deux vérifiées ?
Ou l'une des deux au moins ? etc.
Nous disposons pour cela d'une batterie de mots de « logique », qui permettent de combiner plusieurs conditions en une seule, ou de prendre le contraire d'une condition.
Ces mots ayant aussi d'autres usages, nous nous bornerons pour le moment à expliquer leur fonctionnement en termes de « Vrai » ou « Faux ».

Les tables de vérité, ou de décision... Vous connaissez ? C'est exactement ça.

NOT donne le contraire d'une condition. C'est en français le « non » logique : La condition « non A » est vraie si A était fausse, et est fausse si A était vraie.

OR correspond au « ou » logique. La condition A ou B est vraie si l'une des deux est vraie. Elle est fausse si les deux le sont.

AND correspond au « et » logique. La condition « A et B » est vraie si les deux sont vraies, et est fausse dès que l'une des deux l'est.

XOR correspond au « ou exclusif ». La condition « A ou exc. B » est vraie si l'une des deux est vraie, mais pas les deux. Elle sera fausse si les deux sont fausses, ou si les deux sont vraies.

Ces mots travaillent exactement comme les opérations arithmétiques, et agissent sur les valeurs de la pile. AND et XOR ne travailleront de manière correcte que si vous vous restreignez à la valeur 1 pour « VRAI » (Voir Manuel de Référence pour plus d'explications).
Utilisons le mot TEST défini au 4-2 :

```
1 TEST VRAI  OK
1 NOT TEST FAUX  OK
1 0 OR TEST VRAI  OK
1 1 XOR TEST FAUX  OK
```

Mais

```
2 4 AND TEST FAUX  OK
```

(Attention !!! Voir remarque ci-dessus)
Pour être encore plus cohérent, vous pourriez créer les mots VRAI et FAUX :

```
: VRAI 1 ;        : FAUX 0 ;
```

Vous pourrez alors écrire :

```
VRAI FAUX AND TEST FAUX  OK
```

Voilà, ça n'est pas plus compliqué que cela. Si cela ne vous semble pas clair, n'hésitez pas à faire de nombreux essais.

Quelques exemples simples :
Le nombre est-il positif ou nul ?

```
: 0> = ( n --- cond)  0< NOT ; ( pas strictement négatif)
```

Le nombre est-il entre 12 et 20 (12 et 20 non compris) ?

```
: )12-20( ( n --- cond)
        DUP ( Il y a 2 comparaisons à faire)
        12 >
        SWAP ( Ramène n au sommet de la pile)
        20 <
        AND ;
```

Je cherche une boîte pour mettre mon ordinateur...
(dimensions en cm : l=22 L=42 h=8)

```
: DIM       ( l L h ---)
        8 >
        ROT 22 >                                (ou SWAP 42 > AND)
        ROT 42 >                                (suivi de          )
        AND AND                                 (   SWAP 22 > AND)
        IF ." Ca rentre !!!"
        ELSE ." Boîte trop petite"
        THEN ;
90 45 6 DIM Boîte trop petite  OK
23 44 9 DIM Ça rentre !!!  OK
```

4-5 Répéter une action

Nous disposons de plusieurs commandes permettant des répétitions ; nous allons ici en aborder deux. Vous trouverez les autres dans le Manuel de Référence.
La répétition la plus simple est une commande du type :

```
REPETER <actions> JUSQU'A <une condition>
```

La condition est donc une condition d'arrêt.

Une telle « structure » s'appelle une *boucle* :
(1) Exécuter <actions>
(2) Tester si la condition est vérifiée
    Si oui, arrêter la répétition
    Sinon, retourner au (1), et ainsi de suite...

Tant que la condition n'est pas vérifiée, la « boucle se referme ». Lorsque la condition est vérifée, on « sort de la boucle ».

En FORTH, il faut bien entendu que le résultat de la condition soit déjà sur la pile pour pouvoir faire le test, de sorte que cette structure se présente sous la forme suivante (elle ne peut intervenir que dans la définition d'un mot, entre : et ;) :

```
...       (Début du mot)
BEGIN     (Pour « marquer » le début des actions à répéter)
    ...
    ... (Actions à répéter. En fin d'action, la condition)
    ... (d'arrêt doit rester sur la pile, pour le test)
    ...
UNTIL     (Teste la condition, et recommence la boucle si la)
          (condition n'est pas vérifiée)
...       (Suite du mot)
```

Attention : BEGIN et UNTIL ne sont utilisables que dans la définition d'un mot.

Exemple :
Définissons un mot ++ qui permette de faire la somme d'un nombre quelconque de nombres non nuls. A l'exécution, on tapera simplement Ø comme premier nombre, pour dire à l'ordinateur « C'est ici que se termine la liste des nombres à ajouter » (Les piles, vous vous souvenez ? Premier entré, dernier sorti...).

Au début de l'exécution, la pile contient donc une suite de nombres, au-dessus d'un zéro. Le plus clair est d'utiliser une variable TOTAL, qui contiendra le résultat.

```
VARIABLE TOTAL
: ++        ( Ø n1 n2 n3  ... nn --  somme dans TOTAL)
            Ø TOTAL !                          ( Au début le total est Ø)
            BEGIN
                DUP                             ( Pour pouvoir tester...)
                TOTAL +!                        ( Ajoute au total partiel)
                Ø=                              ( Était-ce le zéro ???)
            UNTIL                              ( Décide si on répète ou non)
            CR ." Le total est :"
            TOTAL ? ;                          ( Affiche le résultat)
```

Puis

```
Ø 2 3 5 7 8 9 ++
Le total est : 34  OK
Ø 4 5 6 Ø 5 ++
Le total est : 5  OK                                ( Arrêt si Ø)
++
Le total est : 15  OK                          ( Somme de 4, 5 et 6)
```

Autre *exemple*, un peu plus compliqué :

Dans un élevage, il y a deux sortes d'animaux : Des bêtes A, et des bêtes B.

Tous les lundis, chaque bête A mange une bête B.
Tous les vendredis, il meurt une bête A et une bête B sur 10 (S'il y avait 13 bêtes A, il en meurt une. S'il y en avait 29, il en meurt 2).
Toutes les 2 semaines, il naît une bête B par couple (S'il y en a 24, il en naît 12. S'il y en a 33, il en naît 16).
Toutes les 3 semaines, il naît une bête A par couple (comme ci-dessus).
Toutes ces naissances ont lieu le jeudi.
Nous sommes un samedi, et il y a 10 bêtes A et 100 bêtes B. Tous les samedis, on fait le compte des animaux. Dans combien de semaines y aura-t-il plus de bêtes A que de bêtes B ?

Cet énoncé semble complexe, mais est très simple à traiter en FORTH. Il suffit d'être un peu méthodique.

Nous aurons besoin de connaître à tout instant le nombre de bêtes de chaque sorte, et la date. Le plus simple est de les définir comme variables.

```
VARIABLE BETES-A
VARIABLE BETES-B
VARIABLE SEMAINE
```

Maintenant, nous allons demander à l'ordinateur de suivre l'évolution de l'élevage semaine par semaine, comme nous le ferions avec un crayon. Mais il ira certainement plus vite que nous...

Chaque semaine, nous aurons :

Le lundi, le repas des bêtes A.
Le jeudi, les naissances.
Le vendredi, les morts.
Le samedi, la comparaison des nombres de bêtes A et de bêtes B.

Nous suivrons l'évolution jusqu'à ce que le nombre de bêtes A soit supérieur au nombre de bêtes B...

Notre test de fin de boucle sera donc :

```
: TEST     ( --- condition)
      BETES-A @ BETES-B @ > ;
```

Et notre boucle se présentera sous la forme générale :

```
REPETER
      repas du lundi
      naissances du jeudi
      morts du vendredi
      recensement des bêtes
JUSQU'A <test positif>
```

C'est-à-dire en FORTH :

```
... BEGIN
   REPAS
   NAISSANCES
   MORTS
   TEST
   UNTIL...
```

Il ne nous reste plus qu'à définir les différentes étapes.

Repas
Soustraire du nombre de bêtes B le nombre de bêtes A.

```
: REPAS          ( ---)
        BETES-B @ BETES-A @ -
        BETES-B ! ;
```

Ou encore (Ajouter l'opposé du nombre de bêtes A)

```
: REPAS          ( ---)
        BETES-A @ NEGATE BETES-B +! ;
```

Morts
Le problème est le même pour les deux, commençons donc par définir un mot commun qui enlève 1 animal sur 10 dans l'une des variables.

```
: MOURIR         ( adr ---)
        DUP  ( pour savoir où remettre le résultat)
        @      ( nombre de bêtes)
        10  /  NEGATE      ( opposé du nombre de morts)
        SWAP  +! ;          ( ajoute à la variable)
```

Puis :

```
: MORTS          ( ---)
        BETES-A MOURIR BETES-B MOURIR ;
```

Naissances
Ici encore le problème est analogue pour les deux races. Le nombre de couples est la moitié du nombre de bêtes, d'où :

```
: NAITRE         ( adr ---)
        DUP @ 2/ SWAP  +! ;
```

(Explications identiques à celles du mot MOURIR)
Les naissances auront lieu pour les bêtes B si le nombre de semaines est pair et pour les bêtes A si le nombre de semaines est multiple de 3.

```
: NAISSANCES  ( ---)
        SEMAINE @      ( Quelle semaine sommes-nous ?)
        DUP 2 MOD 0=              ( Est-ce une semaine paire ?)
        IF BETES-B NAITRE  ( Si oui, naissances de bêtes B)
        THEN
        3 MOD 0=                ( Semaine multiple de 3 ??)
        IF BETES-A NAITRE  ( Si oui, naissances de bêtes A)
        THEN ;
```

Début
Nous sommes un samedi, et nous voulons que le prochain soit le samedi de la première semaine, ce qui revient à dire que nous sommes à la semaine 0.

```
: DEBUT          ( ---)
        0 SEMAINE !
        10 BETES-A ! ( Sans commentaire...)
        100 BETES-B ! ;
```

Fin

En fin d'évolution, nous allons afficher le numéro de la semaine et les effectifs des deux races :

```
: FIN           ( --)
        CR CR ." Semaine :" SEMAINE ?
        CR 15 SPACES BETES-A ? ." bêtes de la race A"
        CR 15 SPACES BETES-B ? ." bêtes de la race B" CR ;
```

Récapitulation

Notre application s'écrira finalement :

```
: EVOLUTION  ( --)
        DEBUT           ( A l'aube du premier lundi)
        BEGIN
            1 SEMAINE +!        ( Semaine suivante)
            REPAS               ( Lundi...)
            NAISSANCES          ( Jeudi...)
            MORTS               ( Vendredi...)
            TEST                ( Samedi...)
        UNTIL           ( Arrêter si test positif)
        FIN ;           ( Afficher le résultat)
```

Voilà, il vous suffit maintenant de taper EVOLUTION...

Quoi de plus simple ? Notez comme FORTH se prête au découpage des gros problèmes en petits modules significatifs...

Sans oublier la possibilité de tester ces modules individuellement !!!

Pour tester le mot NAISSANCES, par exemple :

```
6 SEMAINE ! 33 BETES-A ! 60 BETES-B ! NAISSANCES  OK
BETES-A ? BETES-B ? 49 90  OK
```

4-6 Utilisation de BEGIN...UNTIL

En principe, BEGIN...UNTIL s'utilise si le nombre de répétitions à effectuer n'est pas prévisible, ou si on n'a pas envie de le prévoir. Quand le nombre de répétitions est connu à l'avance, un autre type de boucle est préférable (voir § 4-7).

Il est de bon ton de s'assurer que la boucle a vraiment une fin. Par exemple,

```
: SCIE BEGIN BELL 0 UNTIL ;
```

La boucle n'aura jamais de fin, votre seule ressource sera d'appuyer le bouton d'initialisation...

L'autre usage fréquent de BEGIN...UNTIL est d'arrêter une boucle sur une intervention extérieure, par exemple lorsqu'on veut mettre en œuvre une application répétitive, qu'on puisse stopper en tapant une touche du clavier, ou avec le crayon optique...

Nous disposons à cet effet des trois mots suivants, très souvent utilisés :

?TERMINAL (--- cond)
PTRIG (--- cond)
BUTTON (n --- cond)

?TERMINAL teste si une touche du clavier est enfoncée, et dépose sur la pile la valeur 1 (condition réalisée) si une touche est enfoncée, la valeur Ø (condition non réalisée) dans le cas contraire.

PTRIG effectue la même action, mais teste le bouton qui se trouve à l'extrémité du crayon lumineux. Peu importe que ce bouton soit ou non appuyé sur l'écran, il suffit qu'il soit enfoncé pour que la valeur déposée sur la pile soit 1 (Vous pouvez très bien appuyer sur l'extrémité du crayon optique avec votre doigt).

BUTTON réalise la même action, mais teste les manettes de jeu (si vous en avez).
Comme il y en a deux, numérotées Ø et 1, deux tests sont possibles.

Ø BUTTON teste le bouton de la manette Ø.
1 BUTTON teste le bouton de la manette 1.

Exemple :
Afficher les carrés des nombres entiers, jusqu'à ce que vous décidiez d'arrêter...

```
: RONDE-DE-CARRES              ( --)
        Ø       ( Le premier nombre...)
        BEGIN
            1+                 ( Nombre suivant)
            DUP DDUP           ( Il en faut quatre exemplaires)
            CR .               ( Affiche le nombre)
            * 5 SPACES .       ( Et son carré)
            ?TERMINAL          ( Y a-t-il une touche enfoncée ?)
        UNTIL                  ( Si oui, arrêter, sinon on continue)
        CR ;
```

RONDE-DE-CARRES vous affichera la table des carrés, jusqu'à ce que vous tapiez une touche au clavier.
On peut remplacer ?TERMINAL par PTRIG, par 1 BUTTON ou par Ø BUTTON. Dans ce cas, RONDE-DE-CARRES s'arrêtera alors si vous appuyez le bouton du crayon optique ou d'une des manettes de jeu.

4-7 Répéter une action un certain nombre de fois

Le mieux est de prendre un exemple. Décidons d'afficher les nombres entiers de 1 à 1Ø0 (pas un de plus, pas un de moins).
Si vous tapez au clavier 1 2 3 4 5 6... 1Ø0, vous avez certainement intérêt à relire le paragraphe précédent...

Pour le moment, la méthode consiste pour nous à utiliser BEGIN...UNTIL.

```
: 100-NOMBRES-VERSION-1          ( ---)
        0                        ( Initialise la boucle)
        BEGIN                    ( Début de boucle)
            1+ DUP .             ( Affiche le nombre suivant)
            DUP 100 =            ( Sommes-nous arrivés à 100 ?)
        UNTIL                    ( Si oui, arrêter, sinon recommencer)
        DROP ;                   ( Reste le nombre 100 sur la pile)
```

Ou :

```
: 100-NOMBRES-VERSION-2          ( ---)
        1                        ( Autre initialisation)
        BEGIN
            DUP .                ( Affiche le nombre)
            1+                   ( Au suivant !!!)
            DUP 100 >            ( Sauf s'il est supérieur à 100)
        UNTIL
        DROP ;
```

Ce genre de boucle est très fréquent, aussi y a-t-il des mots spéciaux pour la traiter. Ce sont les mots **DO** et **LOOP**, qui, comme BEGIN et UNTIL, vont par deux.

Notre exemple s'écrit plus simplement :

```
: 100-NOMBRES-VERSION-3          ( ---)
        101 1 DO
              I .
              LOOP ;
```

Quelques explications :

DO Marque le début de la boucle (analogue à BEGIN).
La différence essentielle est que DO met de côté les valeurs 1 et 101, qui seront utilisées par LOOP.

I (Eh oui, c'est un mot FORTH !!!) s'appelle **l'indice** de la boucle. Au début, il vaut 1, et à chaque tour de boucle il augmente d'une unité. La boucle s'arrêtera dès que la valeur de l'indice sera 101.
I n'est en réalité pas un nombre, bien qu'on l'appelle « indice ». En fait, c'est un mot dont l'action est de recopier sur la pile la valeur de l'indice (au sens usuel) du tour de boucle qui est en train de s'effectuer.

LOOP réalise un processus analogue à celui de l'expression
I + 100 > UNTIL (dans 100-NOMBRES-VERSION-2)
— Ajoute 1 à l'indice de boucle.
— Compare la valeur de l'indice à 101.
— Si l'indice est inférieur à 101, nouveau tour...
(FORTH retourne au mot qui suit DO).
— Dès que l'indice est égal à 101, pas d'action apparente pour nous...
(FORTH passe au mot qui suit LOOP).

Les nombres 1Ø1 et 1 marquent les « extrémités de la boucle.
1 est la valeur de début (comme dans la version 2).
1Ø1 est la valeur à partir de laquelle on quitte la boucle.
(la boucle s'arrête dès que l'indice est supérieur ou égal à 1Ø1).

Pièges à éviter

— DO n'est utilisable que dans la définition d'un mot.

— Avant le DO, attention à l'ordre des termes. La valeur de l'indice en début de boucle est la deuxième valeur à écrire.

— Attention à la valeur terminale, n'oubliez pas d'ajoutez 1 à la valeur maximum que vous souhaitez pour l'indice. Par exemple, pour faire varier l'indice de 55 à 79, les valeurs à mettre sur la pile avant DO sont d'abord 8Ø (79 + 1), puis 55.

— Une boucle DO...LOOP est toujours exécutée au moins une fois (comme d'ailleurs BEGIN...UNTIL). Par exemple, "3 19 DO I . LOOP" affichera 3 OK.
L'explication est simple, c'est LOOP qui décide de la fin de la boucle.

— Les valeurs possibles de l'indice vont de −32768 à 32767. Une boucle du genre "4ØØØØ 1ØØØØ DO...LOOP" ne fonctionnerait pas correctement (elle s'effectuerait une seule fois, pour l'indice 1ØØØØ, car pour l'ordinateur, 4ØØØØ est négatif. Voir chapitre 10).

— DO et LOOP sont inséparables. On oublie rarement DO, mais il est fréquent d'oublier LOOP. Dans ces cas, FORTH envoie le message :
? UF ERROR (Unfinished, c'est-à-dire structure incomplète).

Exemples :
On a cambriolé l'épicier du chapitre 3 (voir exercice du § 3-3)

```
: RESERVE-A-ZERO          ( --)
      5 Ø          ( Rubriques Ø à 4 du tableau RESERVE-CONSERVES)
      DO
         Ø I RESERVE-CONSERVES !        ( Met l'article I à Ø)
      LOOP ;
```

Quelques idées :

```
: ETOILES       ( n --)
          Ø DO ." *" LOOP ;          (affiche n étoiles)
: BIPS          ( n --)
          Ø DO BELL LOOP ;           (fait n bips)

5 ETOILES *****  OK
7 BIPS ...
```

Si vous aimez, 25ØØØ BIPS...
Et ici, pas de moyen d'arrêter la boucle, il faut attendre la fin...
(ou utiliser le bouton « INITIALISATION PROGRAMME »)

Ou encore :

```
: ATTENDRE ( n ---)
        0 DO LOOP ; ( n fois « ne RIEN faire » !!!)
```

Par exemple :

```
BELL 1000 ATTENDRE BELL ...
```

Vous pouvez par exemple faire du morse, avec :

```
: TI BELL 1000 ATTENDRE ;
: TA BELL 3000 ATTENDRE ;
```

Puis :

TI TI TI TA TI TA TA TA TI TI TI TA TI TA TI TA TA etc.

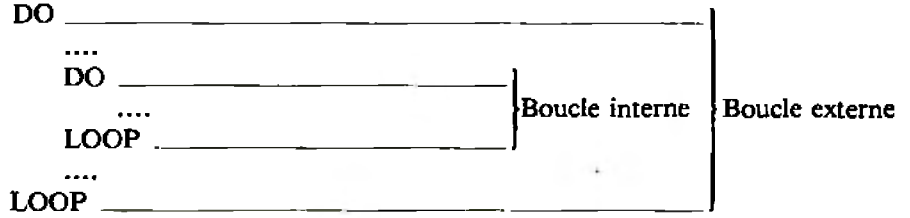
4-8 Boucles imbriquées

Comme pour IF...ELSE...THEN, il est possible d'imbriquer des boucles, lorsque l'action à répéter (on parle de boucle externe) comporte elle-même une répétition (boucle interne).
Tous les schémas sont possibles :

```
   (1)          (2)          (3)          (4)
BEGIN        BEGIN          DO           DO
  ...          ...          ...          ...
  BEGIN        DO           BEGIN        DO
    ...          ...          ...          ...
  UNTIL        LOOP         UNTIL        LOOP
  ...          ...          ...          ...
UNTIL        UNTIL          LOOP         LOOP
```

Faites bien attention, c'est comme pour IF...THEN, chaque UNTIL (ou LOOP) correspond au dernier BEGIN (ou DO) qui n'a pas été terminé par un autre UNTIL (ou LOOP).
Par exemple, dans le schéma (1), le premier UNTIL (fin de la boucle interne) correspondra au deuxième BEGIN (début de la boucle interne...)
Vous voyez l'intérêt des indentations...
De la même façon, une structure du genre ...DO...BEGIN...LOOP...UNTIL n'aurait pas de sens... D'ailleurs, FORTH la refuserait en vous répondant « CD ERROR »...

Dans le cas de boucles DO...LOOP imbriquées (schéma 4), le mot I donne toujours l'indice de la dernière boucle ouverte. Si, à l'intérieur de la boucle interne, vous voulez faire référence à l'indice de la boucle externe, vous pouvez utiliser le mot **J**.



Exemple :
Affichons tous les indices...

```
: VOIR-INDICES          (n1 n2 ---)
          Ø DO
               CR ." = = = = = = = = = = = = = = = = = = = = = = = = = ="
               CR ." Une seule boucle ouverte"
               CR ." I =" I.
               CR BELL ." ----------------------------------------------------------"
               DUP Ø                    ( Duplique n1 pour le prochain tour)
               DO
                    CR 1ØØØ ATTENDRE          ( Pour ralentir l'affichage)
                    ." Deux boucles actives"
                    ." J =" J . ." I =" I .
               LOOP
          LOOP
          DROP ;        ( Il restait un exemplaire de n1 sur la pile)
```

Essayez maintenant 5 3 VOIR-INDICES...

Vous voyez que la signification du mot I dépend de sa position par rapport aux DO et aux LOOP. Lorsqu'il y a deux boucles actives, c'est J qui donne la valeur qu'avait I hors de la boucle interne...

Un autre *exemple* : voulez-vous dessiner un damier ?
Une des méthodes est de numéroter les lignes et les colonnes, et de remarquer que la couleur des cases dépend de la parité de la somme des deux numéros (ligne + colonne) :

```
: BLANC         ( ---) SPACE ;
: NOIR          ( ---) INVCOLOR BLANC INVCOLOR ;

: PARITE        ( x y --- cond x, y = numéros de ligne et colonne)
      + 2 MOD ; ( Calcule la parité de la case. Résultat Ø ou 1)

: PAVE          ( cond ---)
          IF BLANC                              ( Case « impaire »)
          ELSE NOIR                             ( Case « paire »)
          THEN ;
: DAMIER        (---)
               CR ."  Ø123456789"       ( Affiche les numéros des colonnes)
                    1Ø Ø DO                  ( Une seule boucle active...)
                      CR I .                    ( I = numéro de ligne)
                      1Ø Ø DO          ( Boucle interne : dessin d'une ligne)
                    I J PARITE                  ( J = numéro de ligne)
                      PAVE                    ( I = numéro de colonne)
                      LOOP
                    LOOP CR ;
```

En fait, il n'est pas fréquent d'avoir vraiment besoin d'utiliser des boucles imbriquées. L'application précédente s'écrirait plus clairement (en gardant les mêmes calculs) :

```
: PARITE        ( n1 n2 --- cond)
          OVER + 2 MOD ;
```

```
: LIGNE         ( n --- dessine la ligne n)
          DUP .
          10 0 DO
               I PARITE PAVE
               LOOP
          DROP ;

: DAMIER
          CR." 0123456789"
          100 DO
          CR I LIGNE
          LOOP CR ;
```

Pas de J, plus qu'une seule boucle. Plus facile à lire et moins de risque d'erreur... Etudiez bien ces deux versions pour comprendre l'utilisation des mots I et J.

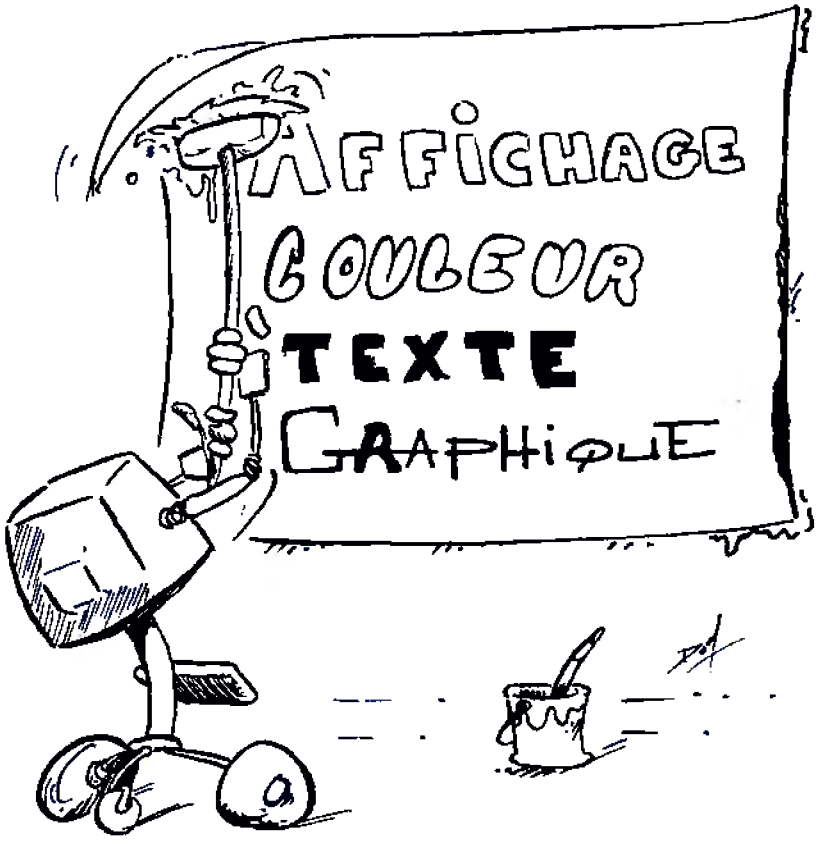
Voilà pour les répétitions. Nous verrons dans la suite de très nombreux exemples d'utilisation, tant de BEGIN...UNTIL que de DO...LOOP.

Si vous voulez avoir une idée de la richesse de FORTH dans le domaine des commandes de structure (conditions ou répétitions), voyez à l'annexe C et dans le Manuel de Référence les mots WHILE, CASE, GODO, et leurs mots annexes.

Par ailleurs, si vous trouvez ces mots insuffisants, FORTH vous permettra (lorsque vous le connaîtrez mieux) de vous en fabriquer d'autres. Mais cela déborde largement de l'objet de ce manuel.

Chapitre 5

L'écran en mode texte



L'ordinateur est très riche en possibilités d'affichage, que ce soit au niveau couleur, texte ou graphique. Nous allons essayer de faire ici un rapide tour d'horizon de l'utilisation en mode texte, c'est-à-dire lorsqu'on décide d'afficher des caractères.

5-1 Généralités : le mode texte

C'est ce mode d'affichage que nous utilisons depuis le début de ce manuel pour écrire à l'écran. Nous avons déjà vu aussi que CLS efface l'écran, que INVCOLOR inverse les couleurs...

5-1-1 La fenêtre écran

En réalité, CLS n'efface que la zone de travail à l'écran, qui pour le moment est l'écran tout entier. On peut modifier cette zone par la commande **WINDOW**, en procédant comme suit :
Votre écran comporte 25 lignes de 40 caractères, qui sont numérotées de Ø à 24.
Si on tape par exemple :
5 12 WINDOW

La zone de travail commencera à la ligne 5, et se terminera à la ligne 12.
Dorénavant, toutes les commandes portant sur l'ensemble de l'écran ne porteront plus que sur cette zone, qu'on appelle la « fenêtre » (WINDOW en anglais)

Vérifications :
CLS n'efface que la fenêtre.
VLIST n'affiche que dans la fenêtre.
Ce qui était écrit en dehors de la fenêtre n'est pas modifié, ce qui peut être bien pratique lorsqu'on veut garder un affichage permanent sur une partie de l'écran, et travailler sur une autre.

Exemple : Table des carrés.

```
: TABCAR          ( n --- Table des n premiers carrés)
        CLS Ø 1 WINDOW          ( La fenêtre comporte 2 lignes)
        ." Nombre    Carré" CR      ( Titres des colonnes)
        ." ----------------"
        2 23 WINDOW             ( Pour ne pas effacer les titres)
        1+ 1 DO
              CR I . 1Ø SPACES I I * .
           LOOP
        Ø 24 WINDOW ;
```

Essayons maintenant
35 TAB CAR...

Vu ? Les titres restent en permanence, le défilement ne se fait que sur les nombres à afficher, dans la fenêtre choisie.

Dans les applications, il sera souvent commode de choisir les fenêtres en faisant un « affichage type » sur un papier, puis de définir des mots du genre :

```
: FENETRE-TITRE-1            Ø 1 WINDOW ;  ( par exemple)
: FENETRE-ADRESSE            3 7 WINDOW ;
: FENETRE-TITRE-2           12 16 WINDOW ;
: FENETRE-COMMENTAIRE       17 24 WINDOW ;
```

Cela évite d'avoir à mémoriser les numéros des lignes des différentes fenêtres. Bien sûr, on choisira souvent des noms plus courts, comme F-T1, F-ADR, etc.

5-1-2 Les options concernant le défilement

Lorsque le texte à afficher arrive à la fin de la dernière ligne de la fenêtre, toutes les lignes déjà écrites remontent d'un cran, et la première ligne est perdue. C'est cet effet qu'on appelle de « défilement » (en anglais « scrolling »).

On peut travailler avec ou sans défilement.
Le mode « **sans défilement** » s'appelle le mode « **page** ». Dans ce cas, il n'y a plus de défilement des lignes lors des affichages.
Dans le mode « **avec défilement** », deux vitesses de défilement sont possibles, le mode défilement normal (celui que vous avez utilisé jusqu'ici), et le mode défilement lent.

La vitesse du défilement se règle par le mot **SCROLL**, qui utilise un paramètre Ø ou 1.

```
Ø SCROLL     Vitesse rapide.
1 SCROLL     Vitesse lente.
```

Exemples :

```
1 SCROLL 35 TABCAR
Ø SCROLL 35 TABCAR
```

On peut arrêter le défilement (se mettre en mode page), en utilisant le mot **PAGE**. Dans ce cas, lorsque la fenêtre est pleine, l'affichage se continue sur la premièr ligne, par dessus ce qui était déjà affiché, sans pour cela effacer complètement la ligne.

Faisons un mot pour bien voir ce qui se passe (Un mot qui affiche des lignes de plus en plus courtes) :

```
: TEST-SCROLL
        39 1 DO CR              ( Pour I = 1 jusqu'à 38)
                40 I - 1        ( Afficher des lignes)
                DO              ( de 39-I tirets ...)
                        ." -"
                LOOP
                        ." *"   ( terminées par une étoile)
        LOOP
        CR ." *" ; ( Puis une seule étoile)

Ø SCROLL TEST-SCROLL
1 SCROLL TEST-SCROLL
PAGE     TEST-SCROLL
(Ecrivez n'importe quoi, vous êtes encore en mode « page »)
Ø SCROLL (pour revenir à un défilement normal...)
```

Note :
— TEST-SCROLL s'écrit plus simplement en utilisant **+LOOP** (Voir Manuel de Référence).

— Le mode « page » sera souvent utile lorsque vous voudrez mélanger texte et dessins, pour éviter de voir vos dessins disparaître à cause du défilement.

— Une erreur fait retourner en mode fenêtre plein écran, avec défilement rapide.

5-1-3 Gestion du curseur

Il n'est pas toujours esthétique de visualiser le curseur, surtout lorsqu'on est en train de dessiner. Il est possible de le faire disparaître ou réapparaître à volonté grâce aux mots :

CSROFF Le curseur n'apparaît pas à l'écran.
CSRON Fait réapparaître le curseur.

Si vous avez tapé CSROFF, le curseur n'est pas affiché, mais il est encore là, et sa position continue bien sûr à dépendre des affichages réalisés. Une erreur provoquera le réallumage du curseur.
Si vous avez besoin de connaître la position exacte du curseur, vous pouvez utiliser les mots **POS** et **CSRLIN** :

CSRLIN (— n) laisse sur la pile le numéro de la ligne où est le curseur.
POS (— n) laisse sur la pile le numéro de la colonne du curseur.
(c'est-à-dire sa position dans la ligne)

Les lignes sont numérotées de 0 à 24, et les colonnes de 0 à 39.

Nous pouvons aussi placer le curseur où nous voulons dans l'écran, en utilisant le mot **LOCATE**.
LOCATE (colonne ligne ---)
place le curseur à l'endroit indiqué.

Exemple :

```
11 15 LOCATE place le curseur en ligne 15, colonne 11.
0 9 LOCATE ." COUCOU" écrira COUCOU au début de la ligne 9.
    (dixième ligne de l'écran, car la première porte le numéro 0)
```

LOCATE ne tient pas compte de la fenêtre. Vous pouvez donc l'utiliser pour écrire hors de la fenêtre où vous êtes en train de travailler. Mais il ne faudra pas oublier, ensuite, de revenir dans cette fenêtre si vous ne voulez pas perturber votre affichage.
Un mot utile pour cela est le mot HOME (à la maison !!!), qui replace le curseur dans le coin supérieur gauche de la fenêtre en cours.
HOME (---) Place le curseur au début de la première ligne de la fenêtre en cours d'utilisation.

Un simple CR ramène aussi le curseur dans la fenêtre (en allant au début de la ligne la plus proche, dans la fenêtre), tout comme CLS (qui efface la fenêtre), ou un autre LOCATE dans la fenêtre...

Exemples :

```
0 5 WINDOW 20 10 LOCATE ." COUCPU" HOME
```

Et pour corriger l'erreur : Le C de COUCPU est en position 20 sur la ligne, donc le P est en position 24. Par conséquent :

```
24 10 LOCATE ." O" HOME
```

ou

```
24 10 LOCATE ." O" CR
```

ou

```
24 10 LOCATE ." O" CLS
```

ou encore

```
24 10 LOCATE ." O" 35 5 LOCATE
```

Essayez ces exemples pour bien voir les différences, et choisir le résultat que vous souhaitez.

5-1-4 La taille des caractères affichés

Nous avons écrit au premier chapitre des mots en très gros caractères, pour voir le fonctionnement et l'« emboîtage » des mots FORTH.
Il y a une méthode beaucoup plus simple, plus rapide et plus performante, c'est d'utiliser le mot SIZE.
SIZE (n1 n2 —) a besoin de 2 paramètres, qui peuvent valoir chacun Ø ou 1.

Ø signifie « taille normale »
1 signifie « taille double »
Le premier paramètre porte sur la hauteur des caractères.
Le deuxième paramètre porte sur leur largeur.
Note : Une erreur provoquera le retour à la taille normale.

Exemple :

```
1 1 SIZE VLIST                  caractères en double taille.
1 Ø SIZE ." MAIGRE !!"          caractères en double hauteur.
Ø 1 SIZE ." TRAPU !!!"          caractères en double largeur.
Ø  Ø  SIZE  ."  OUF !!!!"caracières normaux...
```

5-2 De toutes les couleurs

Comme pour le reste, on peut faire à peu près tout ce qu'on veut de ce point de vue. Mais il y a pour cela quelques grands principes qu'il faut bien comprendre.

5-2-1 Les généralités concernant la gestion des couleurs

Nous pouvons choisir de colorier soit le pourtour de l'écran, soit les caractères que nous écrivons (ou les points lorsque nous ferons des dessins), soit le fond de l'écran.

Chaque emplacement de l'écran a sa propre couleur de fond, et sa propre couleur de caractère (ou de point), et on peut modifier ces couleurs
— soit de manière globale (c'est-à-dire pour toute la fenêtre d'affichage),
— soit de manière locale (caractère par caractère).

Les couleurs disponibles sont les suivantes :
noir, rouge, vert, jaune, bleu, magenta (violet), cyan (bleu clair), et blanc. Elles sont codées dans l'ordinateur par les nombres de 0 à 7, mais pour plus de commodité, elles sont aussi définies par les constantes suivantes :

```
BLACK, RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN, WHITE (--- n).
BLACK   .  0 OK
RED     .  1 OK
GREEN   .  2 OK
YELLOW  .  3 OK
BLUE    .  4 OK
MAGENTA .  5 OK
CYAN    .  6 OK
WHITE   .  7 OK
```

(Note : Si vous possédez un TO 7-70 ou un MO5, 16 couleurs sont alors accessibles. Voir Manuel de Référence. Mots DARK et PALE).

Si vous voulez que vos prochaines attributions de couleur portent sur la couleur du fond, il suffit de prévenir l'ordinateur par le mot **PAPER** (---). Pour qu'ils agissent sur la couleur des caractères, utilisez le mot **INK** (---). Facile à retenir, couleurs du papier et de l'encre...

Si vous voulez que les attributions de couleur portent sur toute la fenêtre, prévenez l'ordinateur par le mot **GLOBAL** (---), et si vous voulez qu'elles ne portent que sur les prochains caractères à afficher (sans changer les couleurs des caractères déjà affichés), utilisez le mot **LOCAL** (---).

Passons à la pratique...

5-2-2 Gestion des couleurs en mode texte

Le pourtour de l'écran peut être coloré par le mot **FRAME** (---). Cette attribution de couleur ne peut bien entendu être que blobale, car il n'est pas possible d'écrire hors des limites de l'écran.

Exemple :

```
RED FRAME (ou encore 1 FRAME)
```

Colorie en rouge le pourtour de l'écran...

Pour colorier le fond ou les caractères dans la fenêtre de travail, le mot est : **COLOR** (n ---).

Il agit sur le fond ou les caractères en fonction du dernier choix réalisé par **PAPER** ou **INK**, et concerne tout l'écran ou seulement les caractères à venir en fonction du dernier choix réalisé par **GLOBAL** ou **LOCAL**.

Exemples :

```
0 24 WINDOW                 ( Pour être sûr de travailler dans tout l'écran).
GLOBAL PAPER WHITE COLOR    ( Le fond devient blanc dans toute la fenêtre).
INK BLACK COLOR             ( Les caractères deviennent noirs).
RED COLOR                   ( Les caractères deviennent rouges. Nous sommes encore en « INK »).
PAPER YELLOW COLOR          ( Le fond devient jaune).
3 20 WINDOW HOME            ( Changeons de fenêtre).
MAGENTA COLOR               ( Le fond devient violet dans la fenêtre).
LOCAL INK BLACK COLOR       « Je suis noir » (« Je suis noir » s'écrit en noir, mais le reste de la fenêtre reste écrit en rouge).
PAPER WHITE COLOR           « sur blanc, c'est plus beau » (Le fond devient blanc, mais seulement pour les nouveaux affichages).
GLOBAL INK BLUE COLOR       ( Tous les caractères de la fenêtre deviennent bleus, mais les couleurs de fond ne sont pas affectées).
```

Voulez-vous voir ce que donnent toutes les combinaisons de couleur possibles entre fond et caractère ?
Avec une couleur de fond fixée :

```
: VOIR-FOND                 ( n ---     n = couleur papier)
        PAPER COLOR         ( Fixe la couleur du fond)
        INK                 ( Dorénavant, COLOR portera sur les caractères)
        8 0  DO             ( Pour couleur de 0 à 7...)
                ! COLOR     ( Fixe la couleur de l'encre)
                 " *"       ( ... ou. « ABCD »...)
             LOOP ;
```

Vous pouvez déjà essayer

```
BLACK VOIR-FOND
RED VOIR-FOND...
```

Si vous tapez WHITE VOIR-FOND, vous resterez avec des caractères blancs sur fond blanc. Il vous faudra taper à l'aveuglette (par exemple RED COLOR) pour voir réapparaître le curseur, et ce que vous êtes en train de taper.
Pour voir tous les arrangements à la fois :

```
: VOIR-COULEURS             ( --)
        8 0 DO              ( Pour couleur de 0 à 7)
                CR I VOIR-FOND
            LOOP
        BLACK COLOR ;       ( Pour ne pas rester en blanc sur fond blanc)
```

Essayez

```
LOCAL VOIR-COULEURS
```

puis :

```
GLOBAL VOIR-COULEURS
```

Vous pouvez aussi essayer ces mots en grande taille de caractère :

```
LOCAL 1 1 SIZE VOIR-COULEURS
GLOBAL VOIR-COULEURS
```

Cette dernière commande est excellente pour la vue, n'est-ce pas ?
En anticipant un peu sur la suite, (chapitre 8), nous pouvons apporter une amélioration sympathique, en insérant le groupe de mots « KEY DROP ». L'effet de ces mots est d'attendre que vous tapiez une touche avant de continuer.
Notre mot devient, en le « spécialisant » pour le mode « couleurs globales »,

```
: VOIR-COULEURS-2          ( --)
        GLOBAL 8 0
        DO CR I VOIR-FOND
                KEY DROP
        LOOP
        BLACK COLOR ;
```

Suivi de :

```
1 1 SIZE VOIR-COULEUR-2
```

Il vous faudra maintenant appuyer 64 fois sur une touche quelconque pour voir défiler une à une toutes les combinaisons possibles de couleur pour toute la fenêtre (que vous pouvez fixer à l'avance comme vous désirez, pour avoir un élément de comparaison avec le reste de l'écran).
Attention, la première combinaison de couleurs qui se présente est « caractères noirs sur fond noir » !!!... Autrement dit, on ne voit rien ! Mais dès que vous taperez une touche, vous vous trouverez en caractères rouges sur fond noir... Tout s'éclaire !!!

Si vous avez souvent à changer à la fois la couleur du fond et celle des caractères, il peut être commode de définir le mot :

```
: COULEURS          ( n1 n2 ---)
        PAPER COLOR INK COLOR ;
```

Pour obtenir des caractèrs bleus sur fond jaune, il vous suffira d'annoncer :

```
GLOBAL BLUE YELLOW COULEURS     (Pour toute la fenêtre)
ou
LOCAL  BLUE YELLOW COULEURS     (Seulement pour les caractères à venir)
```

5-2-3 Autres commandes

INVCOLOR (---) inverse purement et simplement les couleurs des caractères et du fond. La couleur du fond devient couleur des caractères, et vice versa.

```
BLUE YELLOW COULEURS     (caractères bleus sur fond jaune).
```

Puis :

```
INVCOLOR        donne des caractères jaunes sur fond bleu...
```

Tout cela, bien sûr, uniquement dans la fenêtre et en tenant compte du mode global ou local.

Exemple :

```
GLOBAL BLUE YELLOW COULEURS
10 15 WINDOW BLACK GREEN COULEURS
LOCAL 0 24 WINDOW VOIR-COULEURS
GLOBAL INVCOLOR         (Inverse les couleurs pour tout l'écran)
INVCOLOR                (Revient à l'état précédent)
LOCAL INVCOLOR
```

Les caractères affichés dorénavant seront blancs sur fond noir, car la dernière attribution de couleur était en noir sur fond blanc (à la fin du mot VOIR-COULEURS).

On peut en outre décider de conserver les couleurs comme elles sont actuellement, par le mot **COLOROFF** (---). Les affichages suivants respecteront alors les couleurs affectées aux différents emplacements de l'écran, et les modifications locales de couleur seront sans effet, tant que cette option n'aura pas été supprimée par le mot **COLORON** (---).
Les affectations globales de couleur ne sont pas concernées par les mots COLORON et COLOROFF.

Pour bien comprendre ce fonctionnement :

```
: TRANCHES          (---)
            GLOBAL
            0 5 WINDOW BLACK CYAN COULEURS
            6 15 WINDOW RED YELLOW COULEURS
            16 24 WINDOW GREEN MAGENTA COULEURS
            0 24 WINDOW ;
```

Puis :

```
TRANCHES COLOROFF VLIST ...
COLORON VLIST ...
LOCAL
COLOROFF BLUE COLOR ." AZERTYUIOP"
COLORON BLUE COLOR ." QSDFGHJKLM"
```

Avec l'option COLOROFF, seul le texte change, la couleur reste la couleur précédemment attribuée à la position où on écrit.
Avec l'option COLORON, la couleur change en même temps que le texte.

L'option COLOROFF donne une impression de superposition du meilleur effet, on a l'impression que le texte défile sur un transparent devant des projecteurs de couleurs différentes. Essayez par exemple :

```
TRANCHES 1 SCROLL ! 1 SIZE COLOROFF VLIST
```

Et si nous faisions une petite pause, le temps d'essayer une application un peu plus conséquente... Un jeu genre Jackpot, ça vous convient ???

5-3 Un exemple détaillé

Nous allons faire défiler dans une fenêtre 3 caractères pris au hasard parmi 4, et faire en sorte que lorsque vous appuierez sur l'extrémité du crayon optique, le défilement s'arrête et que l'ordinateur vous dise si vous avez gagné (si les 3 caractères sont les mêmes) ou perdu (dans le cas contraire).

D'abord quelques mots de commodité...

```
: COULEURS              ( n1 n2 --        fixer les couleurs)
            PAPER COLOR INK COLOR ;
```

```
: ATTENDRE              ( ---       Temporisation entre les affichages)
          2000 Ø DO LOOP ; ( 2000 fois ne rien faire !!!...)

: ZOU       (Remet l'écran dans son état initial, à la fin du jeu)
          INITSCR BLUE CYAN GLOBAL COULEURS CLS LOCAL ;
```

Nous venons d'utiliser un nouveau mot, **INITSCR**. Son action est, comme son nom l'indique, de remettre les paramètres de l'écran dans leur état initial, sauf les couleurs. La définition exacte de INITSCR est :

```
: INITSCR Ø SCROLL Ø 24 WINDOW CSRON COLORON Ø Ø SIZE ;
```

Nous aurons d'autre part à afficher « gagné » ou « perdu » quelque part dans l'écran...
Par exemple sur la dernière ligne.
Pour pouvoir plus facilement modifier cela ultérieurement, nous allons définir des mots différents suivant les cas.

```
: BRAVO     (à exécuter en cas de gain)
          ." GAGNE" BELL BELL BELL ; ( Cris de joie)

: BOF       ( si on a perdu...)
          ." PERDU" ; (Ici, il vaut mieux se taire)
```

Lorsque nous aurons testé si nous avons gagné ou perdu, nous aurons sur la pile une valeur de vérité, Ø pour « perdu », et 1 pour « gagné ». Nous pouvons donc écrire le mot qui fera l'affichage correspondant :

```
: RESULTAT      (cond ---)
          Ø 24 LOCATE IF BRAVO ELSE BOF THEN ;
```

Un nouveau mot...
Pour choisir le caractère à afficher, nous allons utiliser le hasard (pour imiter un vrai Jackpot, il serait plus correct d'utiliser le mot MOD, afin de simuler un nombre fixe de symboles par roulette).

RND (n1 --- n2)
n2 est un nombre au hasard compris entre Ø et n1-1 (ce qui fait bien n1 valeurs possibles).

Voilà, nous avons tout le matériel nécessaire, il ne nous reste plus qu'à écrire le reste. L'idée est simple, nous allons nous mettre en double taille, et écrire 3 caractères toujours au même endroit de l'écran (lignes 6 et 7, au milieu). Nous répéterons cette action jusqu'à ce que le bouton du crayon optique ait été enfoncé. Alors, nous testerons les trois caractères affichés pour connaître le résultat. Notre application devrait donc se présenter à peu près ainsi :

JEU
Initialisations
  Prévoir l'affichage de l'écran tout entier :
    Effacer l'écran, fixer les couleurs, supprimer le curseur
    Passer en grande taille et afficher un titre.
  Prévoir la fenêtre d'affichage et y mettre d'autres couleurs.
  Se mettre en mode COLOROFF, pour ne pas modifier les couleurs de la fenêtre.

Répéter l'action suivante :
  Afficher 3 caractères au « bon endroit »
  Attendre un peu, puis tester le bouton du crayon optique.
Jusqu'à ce que le bouton soit enfoncé...
Enfin, lorsqu'on a quitté la boucle,
  Comparer les caractères affichés pour voir si on a gagné.
  Afficher le résultat.
  Attendre qu'on appuie de nouveau sur une touche...
    (Pour permettre de lire le résultat)
  Remettre l'écran dans son état initial.
C'est fini...

Il semble raisonnable de définir les 3 mots INITIALISER, LAROUTOURNE, et FIN-JEU. Voyons-les séparément...

```
: INITIALISER     ( —)
      Ø 24 WINDOW BLACK WHITE GLOBAL COULEURS CSROFF CLS
          1 1 SIZE 13 4 LOCATE ." JACKPOT"
      1Ø 11 WINDOW 16 1Ø LOCATE LOCAL
          RED YELLOW COULEURS ." A" ( on écrit un caractère quelconque)
          MAGENTA BLACK COULEURS ." B" ( destiné uniquement à réserver une)
          WHITE BLUE COULEURS ." C" ( position dans la fenêtre...)
      COLOROFF ;
```

Pour le mot LAROUTOURNE, nous aurons besoin de choisir des caractères au hasard. Nous verrons plus tard des moyens plus efficaces, mais nous pouvons d'ores et déjà écrire un mot, disons SYMBOLE, qui écrive un caractère au hasard parmi les 4 caractères « * », « & », « $ », et « ».
Il faut aussi que ce mot laisse un moyen de repérer le caractère affiché, pour pouvoir tester en fin de boucle l'égalité des 3 caractères.
Choisissons de coder nos caractères de la façon suivante :

Ø pour *
1 pour &
2 pour $
3 pour #

Et pour conserver les valeurs, nous pourrions laisser sur la pile les nombres Ø, 1, 2 ou 3, mais il ne faudrait pas oublier de les faire disparaître dans la boucle (sinon la pile serait vite saturée). Il est tout compte fait aussi simple de définir 3 variables, ou un tableau de 3 nombres...

```
3 ARRAY CODES
: SYMBOLE       (n —        n=Ø, 1 ou 2 numéro d'affichage à l'écran)
          4 RND                 (nombre au hasard de Ø à 3)
          DUP Ø=                (Est-ce Ø ?)
          IF ." *"              (Si oui, afficher une étoile)
          ELSE DUP 1 =          (Sinon, est-ce 1)
              IF ." &"          (Si oui, afficher « & »)
              ELSE DUP 2 =      (... Et ainsi de suite)
                  IF ." $"
                  ELSE ." #"
```

```
            THEN
        THEN
    THEN
    SWAP CODES ! ; (mise du code du caractère dans le tableau)

: 3-SYMBOLES    (--- Choix et affichage des 3 caractères)
    0 SYMBOLE 1 SYMBOLE 2 SYMBOLE :
```

Et enfin :

```
: LAROUTOURNE  ( --- Bouche de jeu)
    BEGIN
        16 6 LOCATE 3-SYMBOLES
        ATTENDRE        ( Pour vous laisser le temps de réagir)
        PTRIG           ( A-t-on appuyé sur le light-pen ?)
    UNTIL ;             ( Si oui, on arrête la boucle...)
```

Pour terminer :

```
: FIN-JEU       (---)
    0 CODES @ 1 CODES @ =
    0 CODES @ 2 CODES @ =
    AND                 ( Les 3 codes sont-ils égaux ?)
    RESULTAT            ( Affiche le résultat)
    KEY DROP            ( Attend la frappe d'une touche)
    ZOU ;
```

Le mot de la fin, celui qui nous permettra de jouer sera simplement :

```
: JEU           ( ---)
    INITIALISER
    LAROUTOURNE
    FIN-JEU ;
```

Si le jeu ne vous semble pas assez rapide, vous pouvez toujours changer la temporisation dans ATTENDRE. Vous pouvez même vous chronométrer en utilisant le mot suivant :

```
: MILLISECONDES .       ( n ---)
    0 DO 16 0 DO LOOP LOOP ;        ( Attend environ n millisecondes)
```

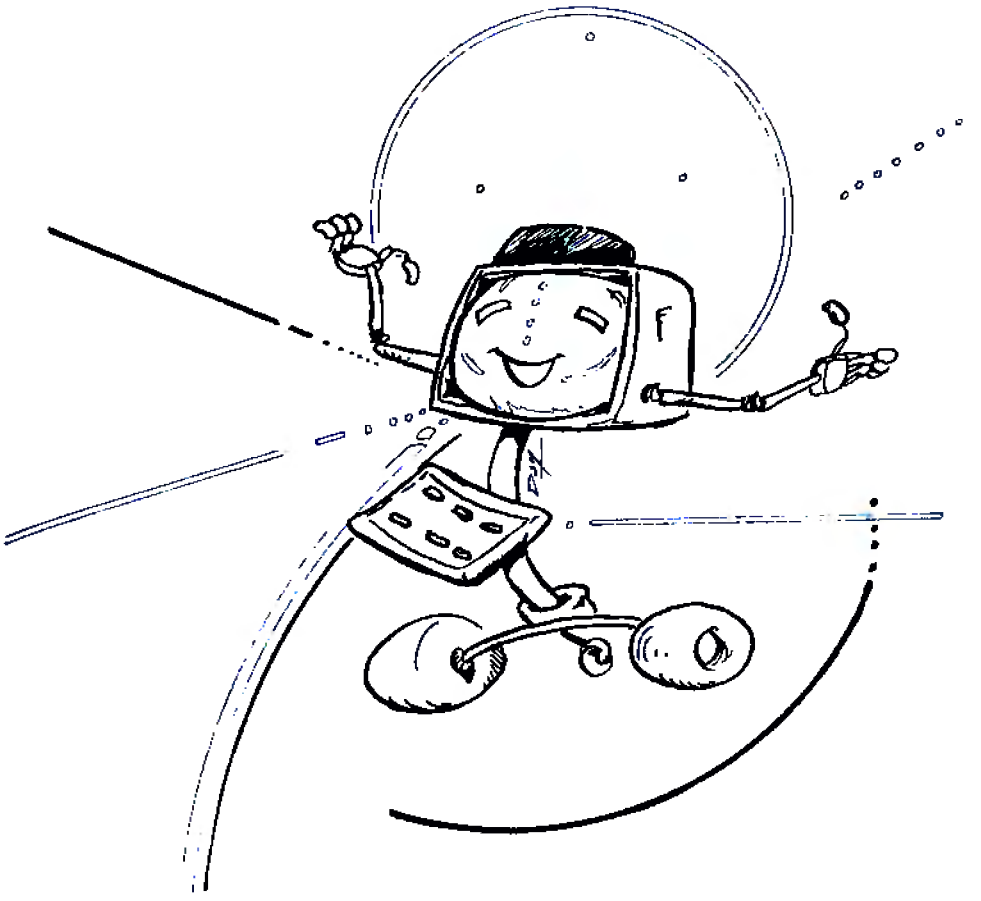
En remplaçant ATTENDRE par « 100 MILLISECONDES », par exemple, vous pourrez avoir une idée de vos réflexes...

Améliorations

Nous verrons au chapitre 6 comment faire un dessin pour décorer le reste de l'écran, au chapitre 7 comment annoncer les résultats en musique, et au chapitre 8 comment fabriquer d'autres dessins pour les caractères qui défilent dans la fenêtre (cartes à jouer, fruits, ou tout ce que vous voudrez...).

Chapitre 6

L'écran en mode graphique



Si vous regardez de près votre écran, vous constaterez que les caractères affichés sont constitués de petits points. Nous pouvons aussi afficher ces points individuellement, ou dessiner des lignes, en noir ou en couleur. Nous verrons aussi dans ce chapitre comment utiliser le crayon optique.

6-1 Des points et des traits

Vous pouvez dessiner des figures composées de points, ou de lignes droites. Nous disposons donc d'un mot pour placer un point, et de mots pour tracer des traits (rectilignes).

Les mots graphiques ne tiennent pas compte de la fenêtre texte. Ils dessineront toujours dans tout l'écran. Toutefois, le défilement agit sur la fenêtre, y compris sur la partie de vos dessins qui s'y trouve.

6-1-1 Un point à la fois

En mode texte, l'écran comporte 25 lignes de 40 caractères. Chaque caractère est écrit dans un carré de 8 points de côté, allumés ou éteints.
Notre écran graphique comportera donc 8 × 25 = 200 lignes de 8 × 40 = 320 points.
Les lignes sont numérotées de 0 à 199, et les colonnes de 0 à 319.
La ligne 0 est celle du haut de l'écran, la colonne 0 est celle de gauche.

Pour mettre un point à l'écran, le mot utilisé est **PSET**.

PSET (colonne ligne ---)

Exemple :
319 0 PSET place un point dans le coin en haut à droite.
Si par hasard vous voyez apparaître un segment, tapez le mot INK. Nous reviendrons sur cette question au paragraphe 6-2. Mais avant de parler des couleurs des points, apprenons d'abord à dessiner des points...

0 0 PSET	place un point dans le coin de l'écran, en haut à gauche.
0 199 PSET	place un point en bas à gauche (toujours dans le coin).
319 199 PSET	Et voilà, nous avons fait les 4 coins...

Nous noterons dorénavant (x,y) les « coordonnées » des points (comme en math...).
x sera le numéro de la colonne, et y le numéro de la ligne.
Par exemple, nous parlerons du point (160,100) pour désigner le point qui se trouve en colonne 160, et en ligne 100.

Attention !!!
Le numéro de colonne **doit** être compris entre 0 et 319, et le numéro de ligne **doit** être compris entre 0 et 199.
Sinon, des choses extrêmement bizarres peuvent se produire... Il se peut même que le bouton INITIALISATION PROGRAMME ne suffise pas à remettre les choses dans l'ordre, et que vous soyez obligé d'arrêter l'unité centrale pour vous en sortir !!!
C'est exactement comme lorsque vous rangez (en utilisant le mot « ! ») une

adresse à une valeur au lieu de faire l'inverse... Si vous écrivez à un endroit anodin, pas de problème, sinon qui sait ce qui peut se passer...
A vous d'y prendre garde...

Exercice :

6.a - Placer un point au milieu de la dernière ligne de l'écran.
- Placer un point au milieu de la dernière colonne de l'écran.
- Placer un point au milieu de l'écran...

Exemple pour matheux :
Dessinons un arc de parabole d'axe la première ligne de l'écran, et de sommet le coin gauche :

```
200 CONSTANT YMAX
100 CONSTANT 2P        (parabole y2=2px)
: PARABOLE      YMAX 0 DO
                          I I 2P */ (y=I . calcul de x)
                          I PSET   (place le point x,y)
                        LOOP ;
```

Les calculs se faisant en nombres entiers (les coordonnées des points sont aussi des nombres entiers), on a intérêt à choisir des grands nombres pour donner une idée correcte de la forme de la courbe.
Noter aussi l'utilisation de */ au lieu de *, puis /. (Voir § 2-10 si vous aviez oublié).
Vous pouvez pour vous entraîner essayer de la tracer à d'autres endroits de l'écran, ou avec un axe vertical...

Exemple :

```
FORGET YMAX
159 CONSTANT YMAX     50 CONSTANT 2P     10 CONSTANT X0
: PARABOLE      YMAX 0 DO
                          I I 2P */ X0 +   ( décalage à droite de X0)
                          YMAX I - PSET  ( pour aller vers le haut)
                        LOOP ;
CLS PARABOLE              ( Ça vous va ???)
```

Ou encore : (Axe vertical)

```
60 CONSTANT XMIN  260 CONSTANT XMAX     160 CONSTANT XAXE
: I->X XAXE - ;
50 CONSTANT 2P       190 CONSTANT YMIN
: PARABOLE      XMAX XMIN
                DO I ( I = abscisse à l'écran)
                   YMIN ( prépare le calcul de d'ordonnée écran)
                   I I->X DUP 2P */ ( ordonnée du point à placer)
                   -                          ( donne l'ordonnée écran)
                   PSET
                LOOP ;
```

6-1-2 Tracer des lignes droites

Pour dessiner un trait allant d'un point à un autre, le mot est **LINE**. Il s'utilise de la manière suivante :
x1 y1 x2 y2 LINE
trace le segment allant du point (x1,y1) au point (x2,y2).

Exemple :
0 0 319 199 LINE Trace une diagonale de l'écran.
319 0 0 199 LINE
ou
0 199 319 0 LINE Trace l'autre diagonale.

Comme pour PSET, il est préférable pour votre tranquillité d'éviter de placer des points en dehors de l'écran (x doit être entre 0 et 319, et y entre 0 et 199).

Quelques autres exemples :

```
: CONE      ( ---)
        WHITE PAPER GLOBAL COLOR INK
        220 70 DO
            I 0                                  ( premier point)
            319 I - 199                          ( deuxième point)
            LINE
            LOOP ;
```

ou encore, exemple voisin (remplissage horizontal) :

```
160 CONSTANT AXE-V                  ( verticale du milieu de l'écran)
100 CONSTANT AXE-H                  ( horizontale du milieu de l'écran)
: SABLIER       ( V---)
        YELLOW GLOBAL PAPER COLOR INK CLS
        AXE-H 0 DO
                    AXE-V I -       ( x1)
                    I AXE-H +       ( y1)
                    AXE-V I +       ( x2)
                    OVER            ( y2)
                    LINE
                LOOP

        AXE-H  0  DO
                    AXE-H I -       ( ligne à partir du milieu...)
                    AXE-V OVER -    ( x1)
                    I               ( y1)
                    ROT AXE-V +     ( x2)
                    I               ( y2... OVER aurait le même effet)
                    LINE
                LOOP ;
```

Exercice :

6.b - Essayez de remplir le sablier de haut en bas, puis de bas en haut.

Un autre mot très commode est **LINETO** (x y —), qui trace un segment (comme LINE) joignant le dernier point placé auparavant au point (x,y).

Exemple :
Pour faire un cadre autour de l'écran (Nous verrons plus loin comment faire plus simplement)
```
0 0 PSET 319 0 LINETO 319 199 LINETO 0 199 LINETO 0 0 LINETO
```

Ou encore, en utilisant LINE pour le premier segment (Le dernier point tracé par LINE est le deuxième point dont on a mis les paramètres sur la pile) :
```
0 0 319 0 LINE 319 199 LINETO 0 199 LINETO 0 0 LINETO
```

Et si maintenant on refait un LINETO, le point de départ du nouveau tracé sera le point de coordonnées (0,0).

Autre exemple :
```
: ENVELOPPE       ( ---)
          100 160 100 100 LINE   150 70  LINETO
          200 100 LINETO         200 160 LINETO
          100 100 LINETO         200 100 LINETO
          100 160 LINETO         200 160 LINETO ;
```

Un conseil
Pour écrire le mot précédent, il est beaucoup plus parlant de procéder en commençant par définir les points qui vont intervenir. Si nous les notons par exemple P1, P2, P3, P4 et P5, cela nous donnera :
```
: P1 100 160 ;        (Dépose sur la pile les coordonnées de P1)
```

Puis :
```
: ENVELOPPE       ( ---)
          P1 P2 LINE             ( Joint P1 et P2)
          P3 LINETO              ( Relie P2 à P3)
          P4 LINETO              ( Relie P3 à P4, et ainsi de suite...)
          P5 LINETO  P2 LINETO  P4 LINETO  P1 LINETO  P5 LINETO ;
```

N'est-ce pas plus lisible ?... Cette écriture met en évidence le trajet suivi :
P1 - P2 - P3 - P4 - P5 - P2 - P4 - P1 - P5

Quel autre langage offre les mêmes possibilités ?

Nous pouvons même faire encore mieux !
```
: ENV-8 LINE 7 0 DO LINETO LOOP ; ( Ligne brisée de 8 segments, 9 sommets)
```

Nous pourrons alors taper (en sens inverse, à cause de l'empilage) :
```
P5 P1 P4 P2 P5 P4 P3 P1 P2 ENV      pour obtenir le dessin précédent.
```

Attention à l'ordre des points P1 et P2 !!! Le premier mot exécuté est LINE, et si on veut que le dernier point tracé par LINE soit relié à P3, il faut que

ses coordonnées soient au sommet de la pile. Il faut donc écrire « P1 P2 » et non « P2 P1 ».

Ou encore :

```
: ENV2 PSET 8 0 DO LINETO LOOP ;
```

Nous pourrons alors taper :

```
P5 P1 P4 P2 P5 P4 P3 P2 P1 ENV2
```

Cela supprime le petit piège dans l'ordre des points P1 et P2, il suffit maintenant d'écrire le parcours vraiment à l'envers.

Exercices :

6.c - Dessinez une étoile à 5 branches (trajet A - B - C - D - E - A). Choisissez vous-même les coordonnées des points A, B, C, D, E et F.

6.d - Améliorations : Faites un mot ETOILE qui vous permette de taper A E D C B A ETOILE et qui dessine l'étoile (ou le pentagone ABCDE), en supposant que vous avez défini les points par des mots, comme ci-dessus.

6.e - Généralisation : Faites un mot qui permette de tracer n'importe quelle ligne brisée, en utilisant comme paramètres la liste de ses sommets, suivie du nombre de côtés.
Exemple : A C B A 3 LIGNE-BRISEE (triangle ABC)
ou même E D C B A 4 LIGNE-BRISEE (ligne non fermée ABCDE)
Suggestion : Le plus clair est certainement de commencer par ROT ROT PSET (plutôt que d'utiliser LINE), puis de faire une boucle.

6.f - Le fin du fin : Si le sujet vous intéresse, essayez d'écrire un mot pour dessiner un polygone fermé, qui ne nécessite pas de répéter le premier sommet comme dans l'exercice précédent. On doit pouvoir taper par exemple :
C B A 3 POLY pour obtenir le triangle ABC (tracé de AB, BC, et CA)
C D B A 4 POLY pour obtenir le quadrilatère ABCD (AB, BC, CD, DA)
Indications : Il suffit de mémoriser les coordonnées de A quelque part, par exemple en utilisant deux variables auxiliaires XA et YA (ou un tableau contenant ces deux nombres).

6-2 Les couleurs en mode graphique

Le mot COLOR peut être utilisé pour fixer la couleur du fond (avec PAPER), mais il n'a pas d'action sur la couleur des points. C'est lui que nous utiliserons pour colorier l'arrière-plan de nos dessins.

En effet, la gestion des couleurs des points est un peu différente de celle des caractères. Les options INK et PAPER signifient qu'on va modifier les couleurs des points ou du fond, mais les options LOCAL et GLOBAL n'ont plus d'incidence sur les couleurs. Les attributions de couleur des points sont toujours locales.

La couleur des points se fixe par le mot **GCOLOR** (n —), qui s'utilise de façon analogue à COLOR.

On peut donc travailler avec des couleurs différentes pour les points et pour le texte.

Exemple :

```
0 24 WINDOW CLS GLOBAL
PAPER WHITE COLOR        ( couleur du fond)
INK GREEN COLOR          ( pour les caractères)
RED GCOLOR               ( s'écrit en vert. Le graphique sera en rouge)
0 0 319 199 LINE         ( dessine un trait rouge en diagonale...)
```

Surprise, certains caractères ont partiellement changé de couleur, juste à côté du trait rouge. Cela tient à la manière qu'a l'ordinateur de gérer les couleurs : Les lignes horizontales sont divisées en segments de 8 points, et chacun de ces segments ne peut recevoir qu'une seule couleur de fond, et une seule couleur de point (lorsque nous écrivons des caractères, cette particularité n'apparaît pas, car les caractères sont justement composés de segments horizontaux de 8 points de largeur, de sorte que leurs couleurs changent « en bloc »...).
De ce fait, si vous fixez une couleur graphique du fond différente de la couleur actuelle du fond, les lignes seront composées de segments de 8 points de côté.

Exemple :

```
CSROFF
0 5 WINDOW GLOBAL PAPER YELLOW COLOR
CLS 6 10 WINDOW RED COLOR                    ( toujours en « fond »)
CLS 13 20 WINDOW GREEN COLOR
CLS 20 24 WINDOW CYAN COLOR
BLACK GCOLOR                                 ( encore en « fond » !)
0 0 100 100 LINE                             ( dessine des carrés noirs...)
300 0 LINETO                                 ( ici, des petits rectangles...)
300 199 LINETO                               ( et là, une bande...)
                    ( Tout cela en 8 points de largeur.)
INK MAGENTA GCOLOR                  ( Passons en couleur « point »)
0 0 100 100 LINE 300 0 LINETO 300 199 LINETO
                    ( Nous avons maintenant de vraies lignes...)
```

Autres exemples :

Un ciel étoilé...

```
: COULEUR          ( ---) 8 RND GCOLOR ;
```

Vous souvenez-vous du mot RND ? Nous l'avons rencontré dans le précédent chapitre (§ 5-3). Il sert à obtenir un nombre au hasard.

```
: ETOILE           ( ---)
          320 RND     (choix d'un x au hasard)
          200 RND     (choix d'un y au hasard)
          PSET ;

: CIEL             ( n ---     n = nombre d'étoiles)
          GLOBAL PAPER BLACK COLOR
          CLS INK                            ( Fond du ciel noir)
          0  DO
                COULEUR ETOILE               ( une étoile quelque part)
             LOOP ;
```

500 CIEL vous donnera un beau ciel, comme vous en avez rarement vu... (Si vous comptez les étoiles, n'oubliez pas les trous noirs !!!)

10000 CIEL nous emmène en plein cœur de la galaxie, dans un fourmillement d'étoiles multicolores... Vous pouvez vous en servir par exemple comme fond pour le jackpot du chapitre précédent...

Des traits au hasard...
Ici un PSET et des tas de LINETO...

COULEUR étant défini comme dans l'exemple précédent,

```
: SOMMET            ( --- x y)
            320 RND 200 RND ; (sans commentaire !!)

: FOND              ( n --- fixe la couleur fond)
            GLOBAL PAPER COLOR CLS INK ;

: LIGNES-HASARD ( ---)
            CSROFF BLACK FOND          ( Les dessins se voient mieux la nuit)
            SOMMET PSET     ( Place un premier point, peu importe la couleur)
            BEGIN
                SOMMET COULEUR LINETO          ( Et un segment, un !)
                PTRIG
            UNTIL (Jusqu'à ce que vous appuyiez sur le bouton du crayon optique)
            KEY DROP (... Pour vous laisser le temps de contempler « notre » œuvre)
            WHITE FOND CSRON ; (... Prêts à écrire ou à dessiner...)
```

Si vous laissez tracer un grand nombre de lignes, vous verrez nettement apparaître les segments horizontaux de 8 points, chaque nouvelle ligne tracée modifiant les voisines.

Pour éliminer les lignes noires (le noir est la couleur 0), il suffit de définir COULEUR par :

```
: COULEUR (---) 7 RND 1+  GCOLOR ;
```

(Prend un nombre au hasard entre 0 et 6 , et ajoute 1)
On pourrait aussi utiliser 8 RND 1 MAX mais ce ne serait plus vraiment un choix de couleurs au hasard, le rouge serait 2 fois plus fréquent que les autres couleurs... Vous pouvez toutefois vous inspirer de ce genre d'idée pour faire des dessins aléatoires comportant une couleur dominante...
Exemple : Dominante jaune (couleur 3). Si votre fond est noir, il suffit de remplacer les points noirs par des points jaunes.

```
: COULEUR-DOM-JAUNE 8 RND DUP 0=                (Le hasard est-il noir ?)
IF DROP 3 THEN                                  (Si oui, remplace noir par jaune)
GCOLOR ;
```

6-3 Des boîtes vides et des boîtes pleines

Nous pouvons dessiner des rectangles à l'écran, grâce au groupe des instructions du type « BOX ». Il y en a quatre :

BOX (x1 y1 x y ---)
BOXF (x1 y1 x y ---)
BOXTO (x y ---)
BOXFTO (x y ---)

BOX dessine un rectangle dont les points (x1,y1) et (x2,y2) forment une diagonale. Le suffixe « TO », dans un des mots précédents, indique que (x1,y1) est implicitement le dernier point précédemment dessiné.
La lettre « F » indique que le rectangle est « plein » (« full » en anglais), c'est-à-dire que l'intérieur sera mis en couleur (fond ou point, suivant le dernier mot positionné).
Dans tous les cas, le dernier point ainsi placé à l'écran sera le point (x,y). Cela est souvent intéressant pour la suite des dessins.

Bien sûr, comme pour tous les mots graphiques, il est important de respecter les intervalles de validité pour les coordonnées des points :
de 0 à 319 pour x, et de 0 à 199 pour y...

Ces mots sont souvent utiles pour encadrer, effacer (par exemple pour effacer un texte ou une partie d'un dessin), ou colorier une partie de l'écran.
Pour effacer du texte, il est en général plus simple d'afficher des espaces, cela évite d'avoir à calculer les coordonnées graphiques de la zone à effacer.

Exemple :
90 20 230 44 BOX entourera le titre « JACKPOT », dans l'exemple du paragraphe 5-3 (dans le mot INITIALISER). A vous de choisir (par GCOLOR) la couleur de ce cadre...
Un autre moyen est de commencer par faire un rectangle plein, mettons en jaune.
Puis, écrire « JACKPOT » en rouge sur fond jaune...
Vous voyez toutes les possibilités qui s'offrent à vous !!!
Les instructions BOXF et BOXFTO vous seront aussi bien utiles pour réaliser des histogrammes, qui ne sont en définitive que des rectangles juxtaposés...

Exemple :
Faisons un histogramme qui représente un tableau de 12 nombres (appelons par exemple ce tableau « FRÉQUENCES »...) :
Bandes de 16 points de largeur. Unité en ordonnée : de 0 % à 100 %

```
32 CONSTANT X0      224 CONSTANT X1          ( 224 = 32 + 12×16)
58 CONSTANT Y0      158 CONSTANT Y1            ( 58 = 158 - 100)
: INIT      (--- prépare l'écran)
      GLOBAL PAPER WHITE COLOR BLACK INK GCOLOR
```

```
    RED COLOR LOCAL CLS PAGE
    10 0 DO
          X0 Y0 I 10 * +
          X1 OVER 10 + BOX
       LOOP
    4 20 LOCATE ." J F M A M J J A S O N D" ;

12 ARRAY FREQUENCES
: HISTOGRAMME       (---)
    INIT PAPER GREEN GCOLOR
    12 0 DO
          I 16 * X0 + Y1                          ( x1, y1 = Y1)
          OVER 15 +                    ( x2... 15 évite les bavures)
          OVER I FREQUENCES @ -          ( y2)
          DOVER DOVER                    ( duplique les 4 coordonées)
          PAPER BOXF                        ( rectangle vert plein...)
          SWAP 1+ SWAP                ( — modifie x2 pour le cadre —)
          INK BOX                          ( ... dans un rectangle noir)
       LOOP
    30 23 LOCATE INITSCR ;                  ( ou KEY DROP ZOU ...)
```

Pour être complet, un mot pour remplir le tableau...

```
: AFFECTER ( n1 n2 n3... n12 ---) 12 0 DO 11 I - FREQUENCES ! LOOP ;
```

Puis, par exemple :

```
30 40 45 60 73 77 43 67 92 55 63 78      AFFECTER      HISTOGRAMME
```

L'avantage de passer par un tableau est de pouvoir facilement le modifier, et de pouvoir le remplir de manière progressive s'il y a beaucoup de données... Modifions les chiffres pour Janvier (0) et Février (1) :

```
91 0 FREQUENCES ! 80 1 FREQUENCES ! HISTOGRAMME
```

Les deux premières bandes de votre histogramme ont changé.

D'autre part, au point de vue graphique, calculez soigneusement les abscisses de vos bandes, si vous voulez éviter que les couleurs bavent (toujours cette histoire des segments de 8 points). En pratique, sauf si vous voulez des histogrammes d'une seule couleur (auquel cas il suffit de les dessiner en couleur « point »), vous serez limités à des bandes de largeur 8, 16, 24, etc. qui commencent aux abscisses 0, 8, 16, etc. et se terminent aux abscisses 7, 15, 23, 31, etc. Essayez par exemple de changer 15 en 16 à la cinquième ligne du mot HISTOGRAMME...

Exercice :

6.g - Faire un mot qui permette d'encadrer une zone de l'écran écrite avec des caractères. Le trait doit passer 3 points au-dessus du haut des caractères du haut, 3 points à gauche du caractère le plus à gauche, etc. Paramètres de la zone texte : haut, bas, gauche, droite (haut et bas entre 0 et 24, gauche et droite entre 0 et 39).

6-4 Le crayon optique

Le crayon optique* (en anglais lightpen) comporte 2 parties :
Un bouton-poussoir, situé à son extrêmité, et un détecteur optique qui permet de viser les points de l'écran. Notons tout de suite qu'il ne détecte pas le rouge ni le noir, pour des raisons techniques (absence de couleur).

Les commandes simples portent sur l'une ou l'autre de ces parties, ou les deux simultanément. A cela s'ajoutent des commandes plus complexes, permettant de repérer des zones de l'écran.

Il y a trois commandes simples :

PTRIG	(--- n)
INPEN	(--- x y)
INPUTPEN	(--- x y)

PTRIG dépose sur la pile une « valeur de vérité » :
Ø si le bouton-poussoir du crayon optique n'est pas enfoncé.
1 si le bouton-poussoir du crayon optique est enfoncé.

INPEN laisse sur la pile les coordonnées du point actuellement visé par le crayon optique. Si la lecture n'est pas valide (visée d'un point hors de l'écran, ou d'une zone noire ou rouge), les valeurs laissées sur la pile sont (−1, −1). Suivant le réglage de la luminosité de votre téléviseur, vous pouvez placer le crayon optique jusqu'à une dizaine de centimètres de l'écran.

INPUTPEN a un effet analogue à celui de INPEN, à cela près que l'exécution s'arrête et attend que le bouton soit enfoncé pour lire les coordonnées du point visé. Autrement dit, si vous avez utilisé INPUTPEN, il faudra appuyer sur l'écran le bouton du crayon optique pour lire les coordonnées du point.

On dit que PTRIG et INPEN réalisent des lectures non bloquantes (en ce sens que la lecture s'effectue sans arrêter l'exécution du mot en cours), tandis que INPUTPEN réalise une lecture bloquante (il bloque momentanément l'exécution, et il faut enfoncer le bouton du crayon optique pour obtenir une lecture)

De plus, si vous possédez un MO5, vous pourrez régler votre crayon optique en exécutant la commande **TUNE** et en pointant alors la barre verticale qui apparaît à l'écran avec votre crayon.

Exemples :
Dessins au crayon optique : Le fond ne peut pas être noir (validité des lectures), et il faut tester la validité des lectures...

(*) Optionnel pour le MO 5

```
: DESSIN        ( n ---)
      GLOBAL PAPER WHITE COLOR CSROFF                ( fond blanc)
      INK GCOLOR CLS                          ( positionne la couleur)
      INPUTPEN                                          ( pour démarrer)
      DUP 0<                                ( la lecture est-elle invalide ?)
          IF DDROP                           ( si oui, arrêter tout de suite)
          ELSE PSET                        ( sinon, on commence la boucle)
              BEGIN
                  INPEN DUP 0<                    ( lecture valide ??????)
                  IF DDROP                         ( sinon, ne rien faire)
                  ELSE LINETO THEN                 ( si oui, un trait, un !)
                  ?TERMINAL
              UNTIL
          THEN CSRON LOCAL ;
```

Puis :

```
GREEN DESSIN
```

Appuyez le crayon optique sur l'écran pour démarrer. Si vous ne visez pas un point de l'écran, le mot s'arrête immédiatement.
Il vous suffit ensuite de promener votre crayon optique devant l'écran pour dessiner en vert.
Pour arrêter, tapez une touche au clavier...
Si vous désirez un dessin avec plusieurs couleurs, nous pouvons modifier facilement ce mot, de manière à ce que lorsque le bouton soit enfoncé, la couleur change (par exemple).
Cela donne :

```
: COULEUR?      ( ---)      7 RND GCOLOR ; ( pas de blanc)
: MODIF?        ( ---)      PTRIG IF COULEUR? THEN ;
```

Et il suffit d'insérer MODIF? dans le mot DESSIN, par exemple juste après BEGIN. Dans ces conditions, il n'est plus nécessaire que DESSIN utilise un paramètre, et le plus radical est de supprimer GCOLOR à la troisième ligne.

INPUTPEN peut servir dans le cas où vous désirez que votre exécution s'arrête momentanément, puis reprenne dès qu'on appuie sur le bouton du crayon optique. Nous avons déjà vu la séquence KEY DROP, qui réalise le même effet, mais attend la frappe d'une touche au clavier. On peut, si on préfère le crayon optique, utiliser la séquence :

```
... INPUTPEN DDROP ... ( DDROP supprime les coordonnées lues)
```

6-5 Conclusion sur le mode graphique

Nous avons fait un tour d'horizon des principales commandes concernant les possibilités graphiques de votre ordinateur en FORTH.
Mais nous sommes loin d'avoir tout abordé ! Nous avons même laissé de côté des commandes très riches de possibilités, que vous pourrez trouver dans le Manuel de Référence. Citons les principales...

PEN

BOXPEN

ONPEN pour définir des zones et tester si le crayon optique pointe dans ces zones (Idéal pour proposer un choix d'options par le crayon optique).

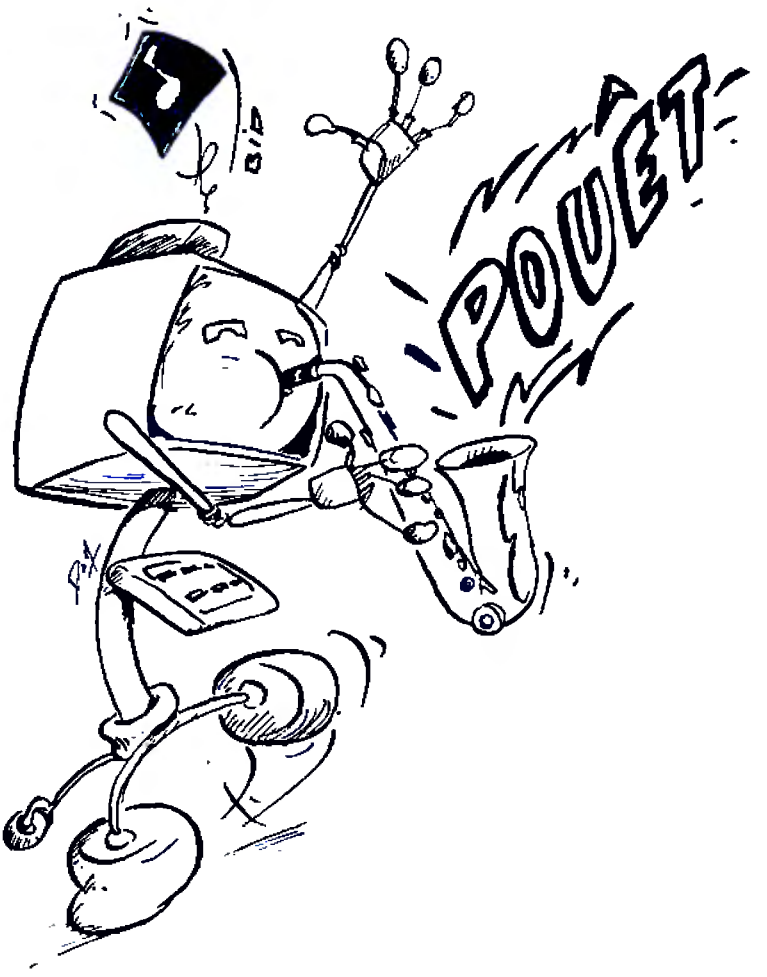
POINT qui donne la couleur d'un point de l'écran, et qui peut même indiquer s'il est tracé en couleur « fond » ou « point ».

BUTTON

STICK si vous avez des manettes de jeux...
Sans oublier la possibilité de définir des caractères graphiques, et de les incorporer dans vos dessins. Cette question sera abordée au chapitre 8.

Chapitre 7

Récréation : musique



Eh oui, nous pouvons aussi faire de la musique !... Nous pouvons jouer 5 octaves chromatiques, faire des silences (!!), choisir notre tempo, et même modifier dans une certaine mesure le timbre.

7-1 Généralités

Nous avons deux façons différentes de procéder, qui correspondent à deux types d'utilisatin différents !
— Écrire une partition, puis la jouer « en bloc » (voir § 7-2).
— Jouer des notes, individuellement, en mode direct (voir § 7-3).

La première méthode sera plus adaptée pour jouer un morceau de musique, alors que la deuxième sera plus pratique lorsque nous voudrons jouer seulement quelques notes, ou faire de la musique aléatoire...
Encore que, bien entendu, chacune des deux méthodes puisse tout faire... Simple question de goût, et de commodité.

Ces deux façons de faire jouer de la musique à votre ordinateur (que nous appellerons, si vous voulez bien, le direct et le play-back) utilisent des mots différents, mais ont toutefois des caractéristiques communes, les paramètres qu'on peut fixer :

L'octave (de 1 à 5)

Valeur par défaut (lorsque vous allumez votre unité centrale) : 4
La valeur 1 donne l'octave la plus grave, la valeur 5 l'octave la plus aiguë.

Le tempo (de 1 à 255)

Valeur par défaut : 5
Lorsqu'on modifie le tempo, on change la vitesse d'exécution, mais les durées relatives des notes sont conservées. Les blanches seront toujours 2 fois plus longues que les noires.
Le tempo le plus rapide correspond à la valeur 1, et le plus lent à la valeur 255.

La durée (de 1 à 96 ... en anglais : duration)

Valeur par défaut : 24
Fixe la durée des notes qui suivent, relativement au tempo.
96 correspond à la durée maximum, celle d'une « ronde ». Une fois choisie cette unité, les durées des notes se déduisent simplement :

Blanche	96/2 = 48	
Noire	48/2 = 24	(valeur à l'initialisation)
Croche	24/2 = 12	
Double croche	12/2 = 6	

Etc. Par exemple, une noire pointée correspondra à une durée de :
24 + 12 = 36

L'enveloppe (de Ø à 255 ... traduit par « pitch »)

Il n'est pas facile d'expliquer l'action de ce paramètre, il joue simultanément sur l'attaque, le maintien,... La valeur Ø correspondra à un son continu, du genre « piano ». Lorsqu'on donne des valeurs plus grandes, le son devient de

plus en plus rapidemment amorti, tout en changeant de sonorité (type « orgue »).
Les valeurs au-delà de 6 ou 7 ne semblent pas indiquées pour faire de la musique, mais peuvent être très intéressantes pour sonoriser des jeux, avec un tempo très rapide...

7-2 La musique en boîte (Play-back)

L'idée est de taper la partition, puis de la jouer. La partition est rangée en mémoire sous un format spécial, appelé « table musicale ». Pour jouer un morceau, on procédera donc en deux étapes bien distinctes :
— Créer une table musicale et la remplir.
— Appeler une table musicale et la jouer.

7-2-1 Créer des tables musicales

Une partition est composée d'une suite de notes, d'altérations (dièse ou bémol), et de commandes de modificatin des paramètres (octave, tempo, durée et enveloppe).
Pour créer une table musicale, il faut d'abord prévoir une taille maximum, en tenant compte des conditions suivantes :
— Chaque note, avec ou sans altération, augmente la longueur de la table d'une unité.
— Chaque modification des paramètres augmente la longueur de 3 unités.
— Les altérations n'augmentent pas la longueur...

Le mot qui crée une table musicale est le mot **MUSTAB**. Il s'utilise de la manière suivante :
nn MUSTAB <nom>

pour créer une table musicale de longueur nn, qui portera le nom <nom>
Par exemple :
100 MUSTAB TAKATITAKITE

Crée une table musicale qui porte le nom TAKATITAKITE, et pouvant comporter un maximum de 100 notes (ou 70 notes et 10 modifications des paramètres musicaux)
200 MUSTAB FUGUE

Crée une table musicale de longueur maximum 200, et de nom FUGUE.

On peut créer autant de tables musicales qu'on le désire, la seule limitatione est la place disponible en mémoire. Ces tables musicales font partie du dictionnaire.

7-2-2 Écrire une partition

Nous avons créé une table musicale, le problème est maintenant de la remplir. Tout d'abord, tapons son nom, pour dire à FORTH quelle table nous allons remplir. Nous pouvons en effet avoir créé plusieurs tables, il convient donc de préciser de laquelle nous allons nous occuper.

Nous disposons ensuite d'une série de mots, dont les noms sont en caractères minuscules **(non accentuées... Attention, le ré ne comporte pas d'accent) :**
do re mi fa sol la si do+ .b .# pause
do, re, mi, fa, sol, la si, do+ placent la note correspondante dans la table, à la suite des précédentes.
pause fait un silence (soupir, pause, etc. suivant la durée des notes).
Ne confondez pas le mot FORTH « pause » avec la pause musicale. Si la durée actuelle est celle d'une « noire », le mot « pause » donnera un soupir musical.

Les paramètres musicaux se fixent entre les notes, et agiront lorsqu'on jouera la table sur les notes (et les silences) qui suivent.
Les mots permettant de fixer ces paramètres sont :

temp	(n ---	n entre 1 et 255)
oct	(n ---	n entre 1 et 5)
dur	(n ---	n entre 1 et 96)
pitch	(n ---	n entre Ø et 255)

Exemple :

```
150 MUSTAB COPAINS (Pour créer la table)
40  MUSTAB POM     (Et une autre...)
COPAINS            (Nous allons remplir la table « COPAINS »)
4 oct 24 dur sol la la .# 36 dur si 24 dur la sol la 48 dur si 3 dur pause
                   (Non, ce n'était pas le radeau)
24 dur la sol la si 5 oct mi fa .# sol 36 dur fa .# 24 dur
                   (de la Méduse, ce bateau...)
mi re 4 oct si la mi la mi la sol si la sol48 dur do 6 dur pause 24 dur mi mi fa .# sol
sol fa .# sol mi sol fa .# mi mi .b si si si 72 dur la 24 dur do+ si la sol sol fa .# sol
mi do si la mi fa .# 96 dur sol
```

Voilà... Ce n'est sans doute pas parfait du point de vue musical.
Quoi qu'il en soit, notre morceau est maintenant prêt à être joué.

7-2-3 Jouer une table musicale

Le mot qui joue une table musicale est **PLAY** (---)
Comme pour l'écriture d'une table, il nécessite qu'on ait auparavant indiqué à quelle table on se réfère.
Ici, c'est inutile, puisque nous avons déjà précisé que nous nous occupons de la table « COPAINS » (à moins que vous n'ayez utilisé une autre table entre-temps...)
Tapons donc :

```
PLAY ...
```

et écoutons...

Tous ces mots sont bien entendu des mots FORTH, et peuvent être utilisés dans d'autres mots...
Par exemple : Rien ne nous empêche de définir les mots
: croche 12 dur ;
: noire 24 dur ;

Puis :
POM (Remplissons cette table...)
3 oct 6 temp 1 pitch
croche do re mi noire fa croche la noire do croche la noire fa
croche pause 0 pitch (demi-soupir)

Enfin :
POM-POM-POM POM PLAY PLAY PLAY ;
POM-POM-POM

Ou même :
```
: POMPOMPOMPOM (---)
      INPUTPEN DDROP POM (Pour commander le démarrage)
      BEGIN
          PLAY
          PTRIG
      UNTIL ;
```

Et vous voilà prêts à utiliser votre crayon optique pour battre la mesure. Tapez POMPOMPOMPOM, prenez votre crayon optique, et appuyez sur l'extrémité. Vous pouvez chanter maintenant, accompagné par votre ordinateur. Lorsque vous aurez fini, il vous suffit d'appuyer de nouveau le bouton du crayon optique.

Petite vérification avant de continuer :
COPAINS PLAY POM PLAY COPAINS PLAY Ça marche !

Autre manipulation intéressante : en utilisant l'éditeur, modifiez les paramètres de POM, pour bien saisir leur effet. Rappelez votre écran, et remplacez (une chose à la fois) 1 pitch par 0 pitch, puis 4 pitch, 3 oct par 1 oct, 5 oct, 6 temp par 1 temp, 5 temp, 20 temp, etc.

Rien ne vous empêche de commencer à écrire une table, de taper PLAY pour essayer ce qui a été entré, puis de continuer à écrire la partition. Les nouvelles notes entrées se mettront à la suite des autres. Attention toutefois à ne pas dépasser la longueur prévue !!! Pour réécrire une table à partir du début, vous pouvez utiliser INITMUS. Il est aussi possible de modifier une table musicale déjà écrite, mais cela sort du cadre de ce manuel. Voir les mots MUSBUF et >MUS dans le Manuel de Référence.

7-2-5 Comment se fatiguer le moins possible...

Lorsque vous voulez entrer une table musicale un peu longue, tout en voulant avoir à réfléchir le moins possible sur les valeurs des paramètres, il sera commode d'utiliser des mots du genre :

```
VARIABLE OCTAVE (pour mémoriser l'octave)
: o!     OCTAVE @ oct ;
: o+     1 OCTAVE +! o! ; (augmente d'une octave)
: o-     -1 OCTAVE +! o! ; (diminue d'une octave)
: *r     96 dur ; (ronde)
: *b     48 dur ; (blanche)
: *n     24 dur ; (noire)
: *c     12 dur ; (croche)
etc.
```

Les autres paramètres changent plus rarement, mieux vaut les laisser hors des tables et les fixer en mode direct, juste avant l'exécution du morceau, surtout en ce qui concerne le tempo...
Mais pour l'octave et la durée, autant laisser FORTH se débrouiller pour fabriquer votre table à votre place !!! (C'est aussi cela, la commodité de FORTH)

Exemple :
Essayons d'écrire la partition de SATIN DOLL (Duke Ellington)
En plus des mots ci-dessus, nous utiliserons :

```
: *bp 72 dur ;          ( blanche pointée)
: *np 36 dur ;          ( noire pointée)
: *cp 16 dur ;          ( croche pointée)
: *cc 8  dur ;          ( double croche)
: p-  *b pause ;        ( demi pause)
: s   *n pause ;        ( soupir)
: s-  *c pause ;        ( demi soupir)
```

Nous avons pris 16 et 8 comme durées des croches pointées et des doubles croches (au lieu de 18 et 6) pour tenir compte des distorsions de durée dans la musique de jazz...

Allons-y...

```
1500 MUSTAB SATIN-DOLL ( Voyons large...)
: THEME-A       ( plusieurs fois utilisé)
    *cp la *cc sol *c la *n sol *np la s- *np la *c sol *np la
    *cp si *cc la *c si *n la *np si s- *np si *c la *np si
    s- *np o+ re *c do *np re s- o- *np si .b *n la .b *c si .b
    *r sol s- ;
```

Pour le moment, notre table est encore vide, le mot THEME-A n'a pas encore été exécuté. Si vous voulez l'essayer, vous pouvez d'ores et déjà taper :

```
SATIN-DOLL THEME-A PLAY INITMUS        (Pour remettre la table à zéro)
```

```
: THEME-B
    *n do+ *cp si .b *cc la *cp sol *cc la *n si .b
    do+ *cp si .b *cc la *cp sol *cc la *c si .b do+ *r do+
    *bp do+ s- *c do+ *cp o+ re *cc do o- *c si la *cp la *cc si *n do+
    o+ *cp re o- *cc do+ *c si la *cp la *cc si *c o+ do re
    *r re *c re *n o- si *cp sol *cc re o+ *cp re o- ;
```

Puis :

```
: PART
    THEME-A sol      THEME-A p- s s- *c sol
    THEME-B          THEME-A ;
```

Et enfin :

```
SATIN-DOLL    4 o!    PART    *n sol s p-    PART    *bp sol s
```

Vérification :

```
10 TEMP PLAY
```

Et si nous ne voulons pas encombrer inutilement notre dictionnaire, nous pouvons faire maintenant :

```
FORGET THEME-A.
```

En effet, nous avons défini la table musicale SATIN-DOLL (et réservé 1500 octets pour elle) **avant** les mots THEME-A, THEME-B, et PART, dont la seule utilité est de remplir la table sans trop nous fatiguer.

```
PLAY
```

Et voilà...

Nous aurions pu faire encore mieux, en définissant notre table musicale avant de définir les mots OCTAVE, o!, o+, *n, ...

Si vous écrivez souvent des partitions, une pratique intéressante est de définir tous ces mots dans un écran de l'éditeur (par exemple l'écran 33), que vous conserverez précieusement sur une cassette ou une disquette (voir annexe relative à l'Éditeur).

Puis, lorsque vous voudrez écrire une partition, vous procéderez de la façon suivante :

— Définir la table musicale. `2000 MUSTAB PARTITION` (par exemple)
— Charger les mots utilitaires. `33 LOAD`
— Écrire votre partition en utilisant ces mots de base... Si un thème revient plusieurs fois, fabriquez un mot intermédiaire comme ci-dessus... Vous pouvez tester ce thème séparément.
— Essayez de jouer la partition pour vérifier le résultat.
— Lorsque vous êtes satisfait du résultat, oubliez les mots inutiles.
(Par exemple : `FORGET OCTAVE`)

Une autre possibilité d'écriture de l'exemple précédent, pour économiser de la place en mémoire, serait de définir plusieurs tables musicales (au lieu d'une grosse) :

```
150 MUSTAB T-A
150 MUSTAB T-B
1   MUSTAB AP1
16  MUSTAB AP2
12  MUSTAB AP3
8   MUSTAB AP4
```

Puis : Définir les mots THEME-A et THEME-B comme ci-dessus...
Ensuite :

```
T-A 4 o! THEME-A
T-B THEME-B
AP1 sol            ( AP comme « après »)
AP2 p- s s- *c
AP3 *n sol s p-
AP4 *bp sol s      ( Il n'y a plus d'après)
```

Puis :

```
: PARTIE T-A PLAY AP1 PLAY T-A PLAY AP2 PLAY T-B PLAY T-A PLAY ;
```

Et enfin :

```
: SATIN-DOLL PARTIE AP3 PLAY PARTIE AP4 PLAY ;
```

Nous jouerons la même chose, mais avec environ 1000 octets de moins !!! La différence est qu'au lieu de taper « SATIN-DOLL PLAY », nous aurons à taper seulement « SATIN-DOLL » (SATIN-DOLL n'est plus ici une table musicale, c'est un mot FORTH qui joue effectivement un morceau de musique).

7-3 La musique en mode direct

Ici aussi, on peut fixer des paramètres et jouer des notes. Mais à la différence des mots précédents, ils seront immédiatement exécutables.

7-3-1 Fixer les paramètres

Les mots concernant les paramètres sont les mêmes que pour les tables musicales, à cela près qu'il est nécessaire de les écrire en majuscules.
Leur action étant immédiate, on peut s'en servir aussi pour modifier les paramètres initiaux lorsqu'on veut jouer une table musicale.

Exemple :

```
2 TEMP 1 PITCH COPAINS PLAY
5 TEMP 0 PITCH COPAINS PLAY      (retour à la « normale »)
```

Il est inutile de modifier les autres, ils sont positionnés dans la table musicale elle-même.

Le mot BELL joue en fait la table « 2 dur do mi sol do+ ». Autrement dit, il sera sensible aux modifications de tempo, d'octave, d'enveloppe, mais pas de durée (il ne modifiera d'ailleurs pas la durée actuellement positionnée).

```
1 OCT 2 PITCH BELL ou 30 TEMP BELL
```

Vous pouvez ainsi personnaliser votre BELL...

Cela soulève une autre question : il vous appartiendra, lors de vos applications, de replacer les paramètres dans leur état antérieur si vous ne voulez pas avoir de surprises. Le problème est le même que pour les couleurs.

7-3-2 Jouer des notes

Ici, la méthode est complètement différente, nous n'avons plus qu'un seul mot à notre disposition, le mot **NOTE**.

NOTE (n ---) joue la note n

n doit être un nombre entre Ø et 12, avec la convention suivante :

Ø NOTE joue un silence (compte tenu de la durée).
1 NOTE joue un do
2 NOTE joue un do dièse
3 NOTE joue un ré
4 NOTE joue un mi bémol
5 NOTE joue un fa

Et ainsi de suite, lorsque n augmente de 1 la note jouée augmente d'un demi-ton.

6 NOTE donne fa dièse
7 NOTE donne sol
...
9 pour la, 11 pour si, et 12 pour le do de l'octave suivant.

Rappelons que les écarts entre deux notes consécutives de la gamme sont d'un ton, sauf les intervalles mi-fa et si-do+ (ou entre le si d'un octave et le do de l'octave suivant), qui sont d'un demi-ton...

Exemples :

Sonoriser une application :

```
: JOIE                ( ---)
      5 OCT 12 Ø
      DO 12 I - DUR          ( durée de plus en plus courte)
         I 1 + NOTE          ( note de plus en plus aiguë)
      LOOP
      2 OCT 24 DUR ;         ( réinitialisation des paramètres)
```

Et

```
: TRISTESSE           ( ---)
      1 OCT 12 Ø
      DO I 2/ 1+ TEMP        ( autre façon de modifier la durée)
         12 I - NOTE         ( note de plus en plus grave)
      LOOP
      2 OCT 5 TEMP ;
```

Vous pouvez maintenant ajouter ces mots dans l'exemple du § 5-3 :

```
: BRAVO ." GAGNE" JOIE ;
: BOF   ." PERDU" TRISTESSE ;
```

Bien sûr, il vous faudra écrire JOIE et TRISTESSE avant de remettre au dictionnaire les mots BRAVO et BOF, puis RESULTAT et FIN-JEU...

Autre exemple de sonorisation : Modifions l'exemple du 6-4...

```
: ZIK                 ( ---)
      5 RND 1+ OCT           ( octave au hasard)
      13 RND NOTE ;          ( et note au hasard)
```

Puis insérez ZIK dans le mot DESSIN, juste après LINETO, et essayez ce nouveau mot DESSIN, en faisant varier les autres paramètres musicaux.

Exemple :

```
2 DUR 1 TEMP 1000 DESSIN ( Très rapide)
5 TEMP 3 DUR 1000 DESSIN ( Plus lent...)
```

Un conseil :

Si vous utilisez souvent les couleurs et la musique, il est commode d'avoir un mot standard qui remette les choses en l'état à la fin de vos applications.

Exemple :

```
: ZOU ( ---) BLACK FRAME GLOBAL PAPER CYAN COLOR
             INK BLACK COLOR RED GCOLOR INITSCREEN CLS
             4 OCT 5 TEMP 24 DUR 0 PITCH ;
```

En ce qui concerne les couleurs, c'est une question de goût...

Et vous pourriez terminer vos mots par :

... KEY DROP ZOU ; ou ... INPUTPEN DROP ZOU ;

KEY ou INPUTPEN servant à vous laisser le temps de contempler l'écran avant de tout effacer...

Exercice :

7.a - Essayez d'écrire un mot qui permette de jouer de la musique au crayon optique, de la façon suivante :
Plus le point visé est en bas de l'écran, plus la note est grave (Vous disposez de 61 notes et de 200 lignes. Utilisez donc 163 lignes, de 0 à 162. Les lignes 0, 1, 2 donneront le do+ de l'octave 5, les lignes 3, 4, 5 le si du même octave, et ainsi de suite, les lignes 160, 181, 182 donnant le do du premier octave).
Plus le point visé est à droite de l'écran, plus la durée est longue. (A vous de faire les calculs).
Pour jouer une note, il faut appuyer le crayon optique sur l'écran, de manière à pouvoir changer de note de façon discontinue (Vous pouvez si vous préférez imaginer le contraire, utiliser INPEN pour jouer et s'arrêter de jouer tant que le bouton du crayon optique est enfoncé... pour permettre de sauter des zones à l'écran en déplaçant le crayon optique tout en le maintenant appuyé).
Enfin, lorsque cela fonctionnera, coloriez l'écran pour faire apparaître les différentes zones (Evitez les couleurs rouge et noire), ou dessinez des traits horizontaux et verticaux, afin de faciliter le jeu.
Lorsque vous aurez fini, il ne vous restera plus qu'à inviter vos amis et leur faire essayer votre mot...

Chapitre 8

Nombres, caractères et codes

Les couleurs, les notes de musique, les points de l'écran, tous les objets que nous avons vus jusqu'ici étaient représentés par des nombres... En fait, l'ordinateur ne manipule que des nombres (et encore, parce que nous voulons bien le voir ainsi, la réalité physique est bien plus complexe...).

L'information élémentaire que peut manipuler un ordinateur est le bit (Ø à 1), mais le microprocesseur qui est au cœur de votre appareil peut traiter 8 bits à la fois (c'est un « 8-bits »). Un tel ensemble de 8 bits s'appelle un **octet**. La mémore de votre unité centrale est composée d'octets, ayant chacun une **adresse**.

Avec ces octets, l'ordinateur peut travailler en base deux avec des nombres allant de Ø à 11111111, c'est-à-dire en décimal avec des nombres de Ø à 255.
En regroupant les octets en « paquets de deux », il faut manipuler les nombres allant de Ø à 65535 (1111111111111111 en base deux). Ce sont nos nombres habituels, ceux que nous avons utilisé jusqu'ici.
En regroupant les octets par paquets de quatre, nous obtiendrons des nombres en « double longueur », pouvant aller de Ø à 4294967295.

Nous allons étudier ici ce que fait l'ordinateur avec 1 octet (8-1), avec 2 octets (8-2), avec 4 octets (8-3).

8-1 En un octet : les caractères et l'ordinateur

Les nombres représentables en un octet vont de Ø à 255. C'est bien peu pour calculer... Par contre, c'est bien suffisant pour coder les symboles dont nous pouvons avoir besoin pour communiquer avec l'ordinateur... Ces symboles sont les lettres de l'alphabet, les chiffres, les signes de ponctuation, et d'autres qui ne sont pas nécessairement liés à un graphisme. On les appelle les **caractères**.

8-1-1 Deux mots fondamentaux : lecture et écriture d'un caractère

Les principaux organes (**périphériques**) qui nous mettent en liaison avec l'unité centrale sont le clavier (pour **entrer** des caractères en tapant des touches), et l'écran (pour afficher des caractères, en « **sortie** »).

En FORTH, les deux mots de base servant à échanger des caractères entre l'unité centrale et les périphériques d'entrée et de sortie sont :

KEY (--- n)
Attend qu'un caractère soit envoyé (fonction bloquante) du périphérique en entrée (exemple : attend la frappe d'une touche) et laisse sur la pile le code du caractère correspondant.

EMIT (n ---)
Envoie au périphérique de sortie (exemple : écrit à l'écran ou sur votre imprimante) le caractère dont le code est n.

Visualisons un peu ces codes :

```
: TEST-CODES CR ( ---)
        BEGIN
                KEY                 ( attend la frappe d'une touche)
                DUP                 ( pour utiliser 2 fois son code)
                CR EMIT ." --"      ( affiche le caractère...)
                                    ( puis affiche le code de la touche tapée)
        PTRIG UNTIL ;
```

Puis :

```
                                TEST-CODES
( tapez un A)                   A--65       ( Le code de « A » est 65)
( tapez un 1)                   1--49       ( Le code de « 1 » est 49)
( tapez un espace)               --32       ( Le code de l'espace est 32)
( tapez la touche [ENTRÉE])      -- 13      ( Le code de la touche ENTRÉE est
                                            13. Affichage bizarre...)
( et le bouton du crayon opti-
que en même temps que la
touche B)                       B--66       OK
```

Rassurez-vous, il n'est pas nécessaire de connaître les codes pour écrire en FORTH, le langage s'en occupe très bien lui-même. Par exemple, EMIT écrit le caractère qu'on a tapé au clavier, nous n'avons pas à nous préoccuper de son code.
Toutefois, comme vous avez pu le voir ci-dessus, les codes ne correspondent pas toujours à des caractères à afficher, mais peuvent correspondre à des actions de l'ordinateur. Par exemple, le code 13 correspond au clavier à la touche ENTRÉE , mais pour l'affichage à un retour au début de la ligne.

Note :
KEY, réalisant une fonction bloquante, stoppe momentanément l'exécution des mots qui l'utilisent. De ce fait, la locution FORTH « KEY DROP » est

très souvent employée pour figer l'écran dans l'attente de la frappe d'une touche. DROP sert uniquement à ne pas perturber la pile.

Exemple :
Traits au hasard.

```
: XY 320 RND 200 RND ;
: TRAITS XY PSET
        BEGIN
            KEY DROP XY LINETO
            PTRIG
        UNTIL ;
```

TRAITS trace un trait à chaque fois que vous tapez une touche.

8-1-2 Le code ASCII

Comme nous l'avons dit, nous disposons de 256 nombres (Ø à 255) pour coder les caractères.
Le code utilisé pour les 128 premiers est standard pour tous les micro-ordinateurs, il s'appelle le « code ASCII ». Seuls quelques caractères (au sens le plus général, pas seulement caractères affichables) varient d'une machine à l'autre.
Vous trouverez en annexe D la liste des codes ASCII et leur action sur l'affichage.
D'une manière générale :
— Les codes standard vont de Ø à 127. Nous verrons au § 8-1-3 l'usage des autres.
— Les caractères affichables correspondent aux codes supérieurs ou égaux à 32.
— Les codes de Ø à 31 s'appellent les « codes de contrôle », et correspondent à des actions de l'ordinateur. Certains de ces codes sont accessibles au clavier de 2 manières différentes, par des touches spécialisées (STOP, RAZ, ENTRÉE, les flèches,...) ou par la frappe simultanée de la touche CNT (Contrôle) et d'une autre touche.

Par exemple : il est équivalent de taper la touche RAZ, ou CNT-L (CNT et L). Le code correspondant est 12, et l'effet immédiat est d'effacer l'écran. En FORTH, on obtient le même résultat en tapant « 12 EMIT »...
Mais attention, si vous tapez la touche RAZ en dehors du mot TEST-CODES, lorsque FORTH attend une commande (et non un caractère) du clavier, l'écran va bien s'effacer, mais FORTH va en plus essayer de comprendre votre entrée comme une commande. Si par exemple vous tapez la touche ENTRÉE immédiatement après avoir tapé la touche RAZ, il va chercher dans son dictionnaire s'il a un mot dont le nom correspondrait au caractère tapé (code 12). Il ne le trouvera pas, et vous aurez donc un message « ? UNKNOWN ».
Alors que « 12 EMIT » est une commande FORTH, et ne provoquera donc pas de message d'erreur.

Certains de ces codes sont définis en FORTH comme constantes :

BL (--- 32) code de l'espace.
BS (--- 8) code de la flèche à gauche (backspace).

D'autres sont intéressants à retenir :

127 « pavé plein »
127 EMIT remplace avantageusement : INVCOLOR ." " INVCOLOR
7 fait un bip, le même que lorsque vous tapez une touche.
14 Passe en mode « TELETEL »... (sur TO 7 seulement)
C'est ce qui vous arrivera si vous tapez CNT-N...
15 Revient en mode « normal » d'affichage.
Si vous avez appuyé par mégarde sur CNT-N, vous pouvez revenir à la normale en tapant CNT-O (qui envoie le code 15 à l'écran).

Donnons deux exemples simples...

Exemple 1 :

Ne trouvez-vous pas que le bruit des touches qu'on enfonce ressemble un peu au tic-tac d'un métronome ? Essayons d'en fabriquer un...
Idée : Faire des « 7 EMIT », séparés par un temps d'attente. Pour rendre notre métronome réglable, il suffit de pouvoir choisir cette attente.
Nous avons vu au § 5-3 le mot ad-hoc, MILLISECONDES.

```
: METRONOME        ( n --)
        BEGIN      DUP MILLISECONDES
                   7 EMIT
                   ?TERMINAL
        UNTIL DROP ;
```

500 METRONOME bat à peu près la demi-seconde.

Exemple 2 :

Le mot SYMBOLE de l'exemple 5-3 s'écrit plus simplement comme ceci :

```
: SYMBOLE        ( n --)
        4 RND
        DUP 35 + EMIT        (caractère de code 35, 36, 37, ou 38)
        SWAP CODES ! ;
```

La différence est que les caractères affichés seront $%& au lieu de *&$ ... Par ailleurs, si vous utilisez des caractères graphiques (voir ci-dessous), cette méthode sera plus agréable, puisque vos codes ASCII pourront être consécutifs.

8-1-3 Extension du mode ASCII. Les caractères graphiques

Dans le FORTH de votre TO 7 ou de votre MO5, le code ASCII a été augmenté pour prendre en compte les voyelles accentuées. Le code 128 donnera par exemple « ç », et les codes suivants les voyelles accentuées (codes de 128 à 148).
Tapez :

```
: ACCENTS          ( --)
            149 128 DO
                              CR I . 3 SPACES 1 EMIT
                    LOOP ;
```

Puis ACCENTS vous évitera de consulter les annexes pour connaître les codes... Pour entrer au clavier une voyelle accentuée (ou le « ç »), il n'y a pas de touche. Il faut taper une séquence de 3 touches commençant par la touche ACC (voir chapitre 1).
Les codes de 149 à 153 donnent aussi des caractères spéciaux (ex : 153 pour ß allemand)

Et les codes au-delà de 153 ?
Ils sont disponibles, et vous pouvez les utiliser avec EMIT. Par exemple : 150 EMIT. Pour le moment, le résultat n'est probablement pas très intéressant, mais il vous est possible de fabriquer vos propres caractères. Le principe est le suivant :
En mode graphique, chaque point de l'écran correspond à un bit de l'ordinateur, qui vaut 1 si le point est de la couleur « point », et Ø si le point est de la couleur « fond ». Les couleurs de point et de fond sont les mêmes pour les huits points d'un même octet, ce qui explique la gestion des couleurs (segments horizontaux de 8 points ne pouvant comporter 2 couleurs différentes, tant en « fond » qu'en « point »). En mode texte, chaque caractère est affiché comme un ensemble de points de l'écran, allumés (de la couleur « caractères ») ou éteints (de la couleur « fond »). Cet ensemble de points se compose de 8 segments horizontaux de 8 points chacun. De sorte qu'un caractère correspond à 8 octets.

Par exemple, lorsque votre ordinateur reçoit l'ordre d'afficher un « A », il « place » à l'écran les segments suivants, l'un en-dessous de l'autre :

```
ØØØØØØØØ
ØØØ11ØØØ
ØØ1ØØ1ØØ
Ø1ØØØØ1Ø
Ø1111111
Ø1ØØØØ1Ø
Ø1ØØØØ1Ø
ØØØØØØØØ
```

Avec la convention suivante :

« 1 » pour « point allumé »
(de la couleur « caractère »)
« Ø » pour « point éteint »
(de la couleur « fond »)

Traduisons ces octets en décimal : nous constatons que l'ordinateur a affiché de haut en bas les segments correspondant aux nombres Ø, 24, 36, 66, 126, 66, 66, Ø.
C'était plus facile en base 2, ne pensez-vous pas ?

Chaque caractère que nous voudrons définir sera de la même façon composé de 8 octets. Pour le fabriquer, il nous suffira de donner le code ASCII que nous désirons lui attribuer, et la liste des 8 nombres correspondants aux 8 segments horizontaux.

8-1-4 Les caractères graphiques « utilisateur »

Mieux, nous pouvons définir ainsi autant de jeux de caractères que nous le souhaitons, chacun de ces jeux de caractères sera rangé dans une table de caractères graphiques, et pourra être utilisé comme les caractères du code ASCII... Chaque table pourra contenir 101 caractères, correspondant aux codes 154 à 255. Ces tables ressemblent beaucoup aux tables musicales, à cela près qu'un élément de la table n'est pas une note, mais se compose de 8 octets.

Passons à la pratique :
On définit une table de caractères par le mot **CHARTAB** (n ---), en procédant comme suit :

n CHARTAB <nom>
crée une table pouvant héberger n caractères
et qu'on pourra appeler par la frappe de <nom>.

Exemple :

26 CHARTAB GREC crée une table qui peut contenir 26 caractères.
(alpha, beta, gamma, etc.)

4 CHARTAB CARTES crée une table pouvant comporter au maximum
4 caractères (trèfle, carreau, cœur, pique)

Une fois ces tables créées, on pourra les sélectionner (soit pour définir les caractères, soit pour les utiliser) en tapant simplement leur nom, exactement comme pour les tables musicales.

Mais avant de nous en servir, apprenons d'abord à les fabriquer...
On sélectionne d'abord la table, puis on écrit les 8 valeurs des octets composant le dessin qu'on veut obtenir (voir ci-dessus), puis le code ASCII désiré, puis le mot **DEFGR**.
Si nous voulions définir un « A » comme premier caractère de la table GREC, nous pourrions taper par exemple :
(En supposant que c'est la table GREC qui est actuellement sélectionnée)
Ø 24 36 66 126 66 66 Ø 154 DEFGR

Nous définirions le caractère 154 comme étant composé des octets dont les valeurs sont de haut en bas Ø, 24, 36, 66, 126, 66, 66, et Ø.
La difficulté est bien sûr de déterminer les 8 nombres Ø, 24, 36, etc.

Il est plus facile d'utiliser la faculté de FORTH de parler dans n'importe quelle base, et de « dessiner » simplement nos 8 segments en base 2. Cela donnera, pour définir un alpha minuscule (pour changer du « A ») :

```
2 BASE !        ØØØØØØØØ
                ØØØØØØØØ
                ØØØØØØØØ
                ØØ11Ø1ØØ
                Ø1ØØ1ØØØ
                Ø1ØØ1ØØØ
                ØØ11Ø11Ø
                ØØØØØØØØ       DECIMAL 154 DEFGR
```

Le voyez-vous, votre alpha ??? Il suffit de cligner un peu des yeux.
Vérification : 154 EMIT. Vous pouvez tester vos caractères au fur et à mesure.
En utilisant l'éditeur, vous pouvez ainsi définir 3 caractères par écran. Si vous préférez travailler en mode direct, utilisez des espaces, pas des flèches !!!...
Il sera bien entendu commode de définir le mot :

```
: BIN     ( ---)     2 BASE ! ;
```

Voilà, ça n'est pas plus compliqué que cela.
Vous pouvez remplir votre table dans n'importe quel ordre, redéfinir un ou plusieurs caractères s'ils ne vous plaisent pas, vous n'êtes pas obligé de terminer votre table, et vous pouvez travailler alternativement sur plusieurs tables...
Quelle souplesse !!!!

La facilité d'utilisation n'est pas moindre : le code à utiliser pour afficher un caractère est celui qui a servi à le définir (quoi de plus simple ?)...
Supposons que nous ayons défini les 2 tables GREC et CARTES. Les premiers caractères de la table GREC sont « alpha » (code 154), « beta » (code 155) et « gamma » (code 156). Les caractères de la table CARTES sont un « trèfle » (154), un « carreau » (155), un « cœur » (156) et un « pique » (157).

```
GREC 154 EMIT 155 EMIT 156 EMIT CARTES 154 EMIT 155 EMIT
```

affichera successivement un alpha, un beta, un gamma, un trèfle et un carreau.
Dans un même affichage, on peut donc, une fois les caractères définis, utiliser des caractères provenant de plusieurs tables.

Les caractères ainsi définis suivent les mêmes attributions de taille ou de couleur que les caractères habituels. Par exemple :
RED INK COLOR CARTES 1 1 SIZE 156 EMIT donnera un gros cœur saignant.

8-1-5 Quelques utilisations des caractères graphiques

1) Supposons que vous ayez entièrement défini vos caractères grecs. Nous pouvons transformer notre clavier en clavier grec :

```
: GREC-KEY              ( --- n)
            KEY DUP         ( code ASCII habituel)
            64 >            ( Ce code est-il entre 65 et 91 ?)
            OVER 91 < AND
            IF              ( Si oui, caractère entre A et Z)
                65 - 154 +      ( modifier le code)
            THEN ;
```

Puis :

```
: CLAVIER-GREC      ( ---)
            GREC
            BEGIN
                GREC-KEY EMIT
            PTRIG UNTIL ;
```

Il ne vous reste plus qu'à taper CLAVIER-GREC pour que les prochaines tou-

ches que vous allez taper s'affichent en grec (seulement les touches alphabétiques majuscules, qui ont un code ASCII entre 65 et 90).

2) Animation

Vous pouvez aussi définir des caractères qui soient des dessins, et les « faire bouger » sur l'écran. Il suffit pour cela de les dessiner, puis de les effacer avant de les redessiner plus loin.

Déplaçons un avion sur la dernière ligne, d'une position n1 à une position n2...

```
1 CHARTAB AVION AVION  HEX 30 98 BA FF BA 98 30 0 DECIMAL 154 DEFGR
: VOLE          (n1 n2 ---)
        OVER 24 LOCATE 154 EMIT          ( Dessine un premier avion)
        KEY DROP                         ( Prêt à partir...)
        SWAP DO
                5 MILLISECONDES          ( Voir § 5-3)
                I 24 LOCATE              ( Se place sur le précédent)
                BL EMIT                  ( l'efface...)
                149 EMIT                 ( et le réécrit plus loin)
                2 NOTE                   ( un peu de bruit)
            LOOP ;
```

Tapons :

```
1 OCT 2 TEMP 6 DUR CLS
```

puis :

```
0 39 AVION VOLE
```

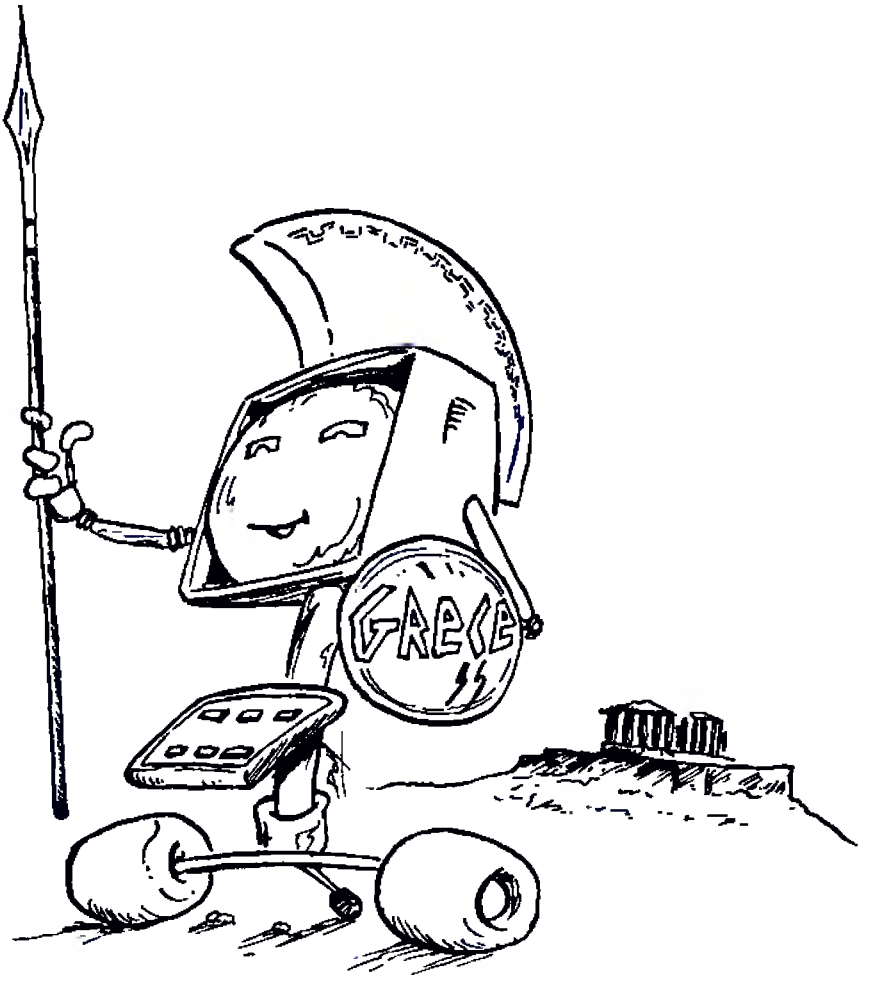
L'avion se place en début de dernière ligne, et dès que nous tapons une touche, il se déplace jusqu'à la fin de la ligne.

On peut accélérer le déplacement ou le ralentir en agissant sur le paramètre de MILLISECONDES (Plus cette valeur sera grande, et plus le « A » se déplacera lentement. On peut aussi faire un mouvement accéléré, en diminuant cette valeur dans la boucle)

Vous voyez quels horizons s'ouvrent à vous !!!

Exercice :

B.a - Créer un mot BZZ qui utilise les paramètres suivants :
VARIABLE TEMPS (Sa valeur est le paramètre de MILLISECONDES)
VARIABLE CAR (Contient le code du caractère à déplacer)
VARIABLE VX VARIABLE VY (Valeurs des déplacements dans chacune des 2 directions, entre l'effacement et l'écriture dans la boucle. Cette pratique permet de se déplacer dans n'importe quelle direction. Par exemple : déplacement vers la gauche si VX vaut −1 et VY vaut 0, déplacement en diagonale vers le haut et vers la droite si VX et VY valent tous les deux 1. Avec VX=2 et VY=2, le mobile ira 2 fois plus vite...).
VARIABLE NB-PAS (Nombre de pas à effectuer)
et qui prenne sur la pile les coordonnées écran du point de départ...
Vous aurez alors un outil vous permettant de déplacer n'importe quel caractère graphique dans n'importe quelle direction, à partir de n'importe quel endroit de l'écran... Il ne vous restera plus qu'à fabriquer des tables d'objets volants (identifiables ou non).



8-2 En deux octets : les nombres standards

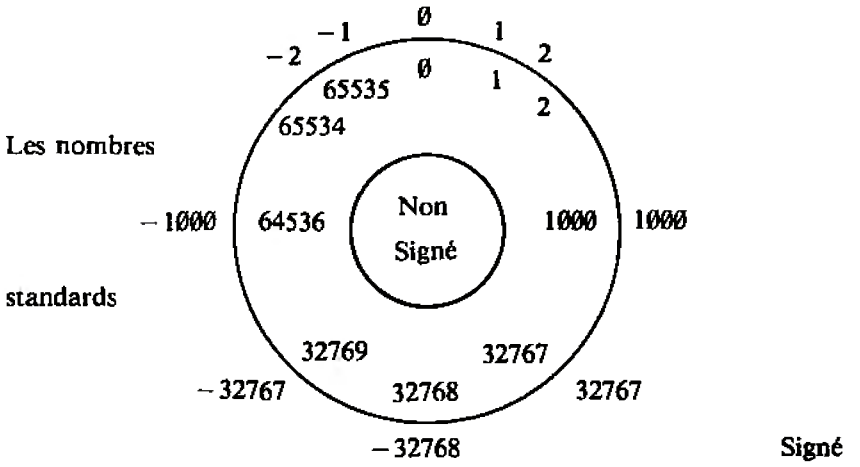
Comme nous l'avons dit, il est possible, en deux octets, de représenter des nombres allant de Ø à 65535 (65536 nombres, compte tenu du zéro).
En réalité, l'ordinateur calcule « modulo 65536 », comme disent les mathématiciens. Cela signifie que pour lui, le nombre qui suit 65535 est Ø
Essayons :

```
65536 . Ø  OK
65534 2+ . Ø  OK
```

Exactement comme sur le compteur kilométrique de votre voiture, le nombre qui suit 99999 est Ø... Dès que la limite de 65536 est atteinte, le compte repart à Ø.
Et cela est vrai pour toutes les opérations...

```
40000 2* . 14464  OK (80000 = 65536 + 14464)
```

L'inconvénient de ce qui précède est que cela ne nous permet de travailler qu'avec des nombres positifs... Qu'à cela ne tienne, si le suivant de 65535 est Ø, c'est que 65535 est synonyme de −1 !!!
De la même manière, 65534 est synonyme de −2, 64536 est synonyme de −1000, et ainsi de suite... (Voir schéma ci-dessous).



On pourrait, en poussant ce jeu jusqu'au bout, dire que −65535 est synonyme de 1. Mais cela n'est pas raisonnable, car ou bien on n'a pas besoin des nombres négatifs ou bien on a besoin des nombres avec un signe (négatif *et* positifs). Si on utilise des nombres **signés** (avec un signe) il est raisonnable qu'il y ait autant de nombres positifs que de nombres négatifs. La notation standard des nombres signés va de −32768 à 32767.
Ces mêmes nombres peuvent être représentés en notation **non signée** (« Unsigned » en anglais), de Ø à 65535.
Cela ne pose aucune difficulté lors des calculs, les seuls problèmes sont liés à l'affichage et aux comparaisons...

Affichage :
FORTH nous propose deux mots, un pour l'affichage signé (le point. que vous connaissez déjà), et U. (affichage non signé).

Exemples :

```
:.U. (n —) DUP .U. ; (Donne les deux affichages)
65535 .U. −1 65535 OK
−1    .U. −1 65535 OK          (pas de problème, il comprend bien).
32767 .U. 32767 32767 OK       (l'affichage est le même)
32768 .U. −32768 32768 OK      (le plus petit nombre signé possible)
```

Ces mots tiennent compte de la variable BASE pour réaliser les affichages.
65533 2 BASE ! U. 1111111111111101 (en base deux...)

Comparaisons :
30000 est-il inférieur à *40000* ? En informatique, la réponse n'est pas si simple, puisque *40000* est aussi −25536 !!! Il faut donc ici aussi des mots de comparaison différents, suivant qu'on travaille en signé ou en non signé. Vous connaissez les mots pour la notation signée (*0*<, *0*> ≒ > et <). Pour les

30000 *40000* <. *0* OK *30000* *40000* U< . 1 OK

Attention aussi lorsque vous utilisez DO...LOOP, l'indice de boucle est considéré comme signé. Autrement dit, « *40000* *30000* DO...LOOP » n'exécute la boucle qu'une fois, avec l'indice *30000*. Puisque *40000* est inférieur à *30000*, la boucle s'arrête tout de suite. Il existe un type de boucle DO spécifique (voir /LOOP au Manuel de Référence).

8-3 En quatre octets : les nombres en double longueur

On distingue, comme pour les nombres standards, des nombres « double longueur » (ou nombres 32 bits) signés et non signés.
Les nombres non signés ont maintenant des valeurs entre *0* et 4294967295.
Les nombres signés, eux, vont de −2147483648 à 2147483647

Les conventions habituelles pour représenter les nombres sur la pile sont :

n pour un nombre standard.
un pour un nombre standard interprété comme non signé.
d pour un nombre double longueur.
ud pour un nombre double longueur interprété comme non signé.

Pour prévenir FORTH que le nombre qu'il lit est un nombre en double longueur, il suffit d'y incorporer un point. Peu importe sa position, il suffit que le nombre contienne un point... Ne confondez pas ce point avec un point décimal du BASIC, le nombre ainsi tapé est bel et bien un nombre entier.

Exemples :
12345. 123.45 1.2345 .12345
donneront tous le même nombre double longueur (qui pourrait être ici un nombre standard), et dont la valeur est 12345 (nombre entier).
Evitez aussi de confondre ce point avec l'ordre d'affichage".". Le point indiquant un nombre double longueur fait partie du nombre, et n'est pas séparé par un espace.
Nous n'insisterons pas ici sur l'utilisation qu'on peut faire de ce point. Il est employé lors des formattages de l'affichage, si vous voulez simuler des nombres décimaux. Vous pouvez consulter à ce sujet votre Manuel de Référence (mots DPL, ,< , etc.).

Les nombres double longueur occupent sur la pile la place de deux nombres standards. En fait, le nombre d s'écrit sous la forme (n1, n2), de sorte que :
d = n1 × 65536 + n2 (n1 étant au-dessus de la pile)
n1 s'appelle la partie de poids fort, n2 la partie de poids faible.

Exemple :
3.45678 .S La pile des données contient 17998 5
Cela signifie que : 345678 = 5 × 65536 + 17998

La plupart des mots concernant les nombres standards ont un équivalent pour les nombres double longueur. Il suffit d'ajouter au nom du mot standard le préfixe « D ».
Les principaux mots concernant les nombres double longueur sont :

Affichage :

D.	(d ---)	Affichage d'un nombre double longueur signé.
UD.	(ud ---)	Même chose, mais interprète ud comme non signé.

Exemple :
2342342.342 D. – 1952624954 OK
2342342342. UD. 2342342342 OK

Manipulations de pile
Nous avons déjà vu les mots concernés au § 2-5, mais nous pouvons maintenant les utiliser autrement... (et justifier leurs noms).

DDUP	(d --- d d)	duplique un nombre 32-bits.
DDROP	(d ---)	supprime un nombre 32-bits.
DSWAP	(d1 d2 --- d2 d1)	échange 2 nombres 32-bits.
DOVER	(d1 d2 --- d1 d2 d1)	analogue à OVER.
DROT	(d1 d2 d3 --- d2 d3 d1)	analogue à ROT.

Variables et constantes

DVARIABLE	Travaille comme VARIABLE, en 32-bits.
DCONSTANT	Travaille comme CONSTANT, en 32-bits.
D@	Travaille comme@.
D!	Travaillle comme !.

Nous ne disposons pas de D?, ni de D, ..., mais vous pouvez très bien les définir.
: D? D@ D. ; : D, , , , ;

Il n'y a pas non plus de tableaux de nombres double-longueur, mais vous pouvez très bien utiliser les tableaux « habituels ». Il suffit de vous souvenir qu'un nombre double longueur occupera la place de 2 nombres standards. Un tableau TABLO de 7 nombres double longueur pourra donc être défini par :
7 2* ARRAY TABLO (ou 14 ARRAY TABLO)

Pour aller chercher le troisième élément du tableau (il est en position 2, vous vous souvenez ?... En FORTH, tout est numéroté à partir de Ø), il faudra taper :
3 2* TABLO D@ (Si votre tableau s'appelle TABLO)

Par ailleurs, il est tout à fait possible de définir *en FORTH* des tableaux de nombres 32-bits, mais cela dépasse le cadre d'un manuel d'initiation. Si la question vous intéresse, voyez CREATE et DOES> au Manuel de Référence...

Opérations arithmétiques

D+	(d1 d2 --- d3)	d3 = d1 + d2
D−	(d1 d2 --- d3)	d3 = d1 − d2
D*	(d1 d2 --- d3)	d3 = d1 × d2
DABS	(d1 --- d2)	d2 est la valeur absolue de d1
DNEGATE	(d1 --- d2)	d2 = −d1
DMIN	(d1 d2 --- d3)	d3 est le plus petit des deux.
DMAX	(d1 d2 --- d3)	d3 est le plus grand des deux.

Comparaisons

Ici encore, les commandes sont inspirées de celles qui concernent les nombres standards, et sont faciles à comprendre.

D0=	(d1 --- cond)	teste si d1 est nul.
D<	(d1 d2 --- cond)	test si d1 est inférieur à d2.
D=	(d1 d2 --- cond)	teste si d1 est égal à d2.

Attention :

<cond> est un nombre standard (en 2 octets), qui vaut 1 si le test est positif, et 0 si le test est négatif. C'est bien normal, puisque le résultat du test est utilisé par IF, ou UNTIL (et d'autres mots que nous verrons au chapitre 12), qui, eux, ont besoin de nombres 16-bits.

8-4 Mots faisant intervenir des nombres standards et double longueur

Voyons d'abord comment passer de l'un à l'autre pour les nombres qui sont sur la pile.

Problème 1 :

Vous avez sur la pile un nombre 32-bits, et vous savez qu'il est entre −32768 et 32767. Vous voulez utiliser ce nombre comme nombre 16-bits.
La solution est simple, il suffit de faire un DROP pour enlever la partie de poids fort.

Problème 2 :

Vous avez un nombre 16-bits, et vous voulez l'utiliser comme nombre double longueur. Il suffit auparavant de le transformer par le mot magique :

S->D (n --- d)
Transforme n en nombre double-longueur

Exemple :

```
123. DROP.          123  OK
 12     S->D D.     12  OK
 33000. DROP .     -32536  OK      (Bien sûr, supérieur à 32767)
 33000. DROP U.     33000  OK
-2000. DROP .      -2000  OK
-2000. DROP U.     63536  OK
```

Ces 2 derniers exemples sont un peu plus compliqués à expliquer...
Nous n'entrerons pas dans le détail (Pour les curieux, l'explication réside dans le « complément à deux », qui est utilisé pour représenter les nombres négatifs).

```
 2345 S     D D.  2345  OK
-456 S      D D. -456  OK
```

Quand un langage utilise plusieurs sortes de nombres, il est bien normal qu'il fournisse aussi le moyen de passer d'une représentation à l'autre, et vice versa...
Ce qui par contre est moins fréquent, c'est que FORTH a aussi prévu que vous pourriez avoir envie de multiplier ou de diviser entre eux des nombres de types différents.
Nous disposons à cet effet de plusieurs opérations mixtes, qui opèrent de la même façon que les opérations sur les nombres 16-bits, mais diffèrent par les types des nombres utilisés ou laissés sur la pile...

M*	(n1 n2 --- d)	multiplication
U*	(un1 un2 --- ud)	
DN*	(d1 n --- d2)	
UDN*	(ud1 un --- ud2)	
M/	(d n1 --- n2 n3)	division (voir /MOD)
U/MOD	(ud un1 --- un2 un3)	
M/MOD	(ud1 un1 --- un2 ud2)	

Cette terminologie n'est pas très satisfaisante, surtout en ce qui concerne les divisions... Mais ces noms sont pratiquement standards.

Par exemple : Nous voulons définir un mot qui calcule « factorielle n », où n est le nombre sur la pile. Connaissez-vous la fonction « factorielle » ?
Pas compliqué :
C'est le produit de tous les nombres de 1 jusqu'à n (pour n entier positif).
EX : factorielle-10 donne 2 × 3 × 4 × 5 × 6 × 7 × 8 × 9 × 10 = 3628800
La première idée est de travailler avec des nombres standards, mais comme vous pouvez le constater les valeurs de cette fonction augmentent très vite... très très vite... (Voyez factorielle-10 !!).
Avec des nombres double longueur, nous irons (un peu) plus loin... Prenons même des nombres non signés, puisque nous ne travaillons qu'avec des nombres positifs.
L'idée est la suivante :
n est sur la pile...
On part de 1 (double longueur non signé)
Pour I variant de 1 à n
    multiplier le résultat par I
Afficher le résultat.

Ce qui donne le mot FACT suivant :

```
: FACT              ( n --- ud)
      1.            ( Met sur la pile le nombre 1 (32-bits non signé)
      ROT           ( Ramène n au-dessus de la pile. Attention, 1.)
                    ( occupe 2 fois 16-bits. C'est donc bien ROT qui)
                    ( convient, et non pas SWAP...)
      1 + 1 DO      ( Pour I allant de 1 à n... Attention au 1+ ...)
            I UDN*      ( La multiplication adéquate...)
        LOOP
      UD. ;         ( Affiche le résultat)
```

Et maintenant :

```
10 FACT 3628800  OK
12 FACT 479001600  OK
13 FACT 1932054504  OK
```

Ce dernier résultat est faux !!! A la main, on trouve :

```
factorielle-13 = 6227020800
```

Après tout, c'est normal. C'est trop grand pour être un nombre en 4 octets. Nous nous sommes un peu fatigués, mais nous pouvons aller jusqu'à 12 ! Alors qu'en 16 bits, même en non signé, nous ne pouvions dépasser 8...

8-5 Entrée et conversion des nombres

Si vous voulez que vos mots demandent un nombre à l'utilisateur, utilisez **LINPUT** ou **INPUT.**
Ces deux mots interpréteront toujours le nombre que vous entrerez comme un nombre double longueur. Si vous voulez utiliser des nombres standards, il vous faudra donc faire un DROP avant d'utiliser ces entrées.

```
INPUT       (  --- d)
LINPUT      (n --- d)
```

La différence entre INPUT et LINPUT est sur le nombre de caractères (donc ici de chiffres) que vous pouvez taper. C'est dire qu'une saisie par INPUT devra se terminer par la touche ENTRÉE , alors que si vous tapez par exemple :

```
3 LINPUT
```

votre utilisateur ne pourra taper un nombre de plus de 3 chiffres.

Exemple :
Table des carrés

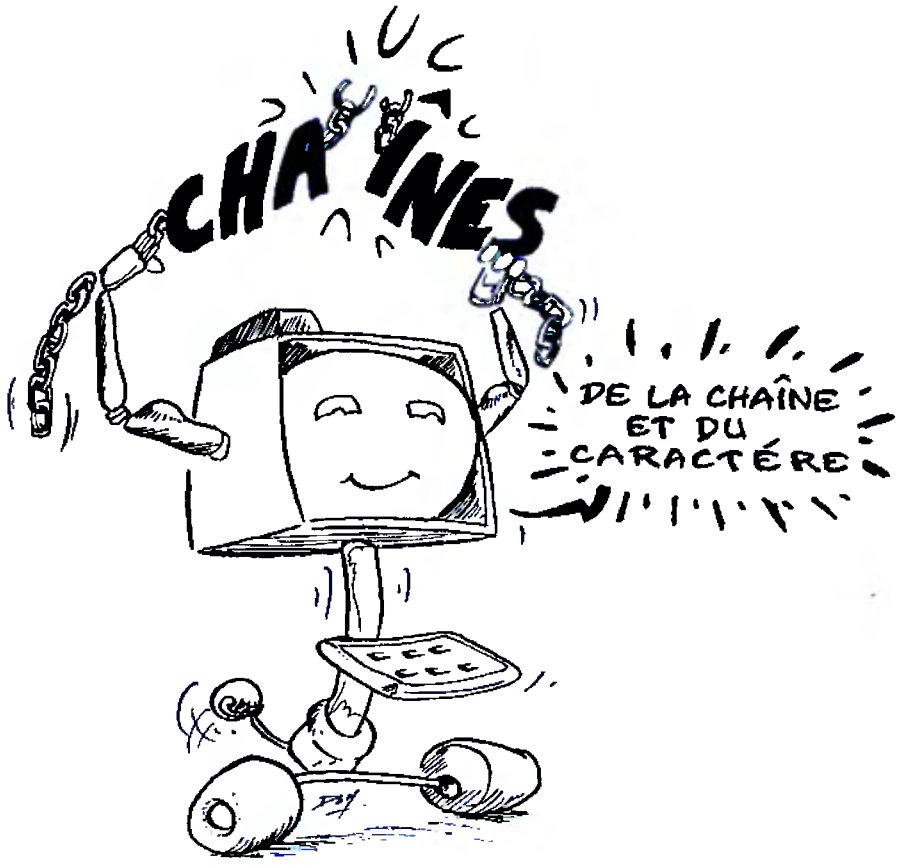
```
: CARRES        ( ---)
          CR ." Ce mot affiche les carrés des nombres"
          CR ." positifs à partir de :"
          INPUT DROP 0 MAX        ( à partir de combien ?)
          CR ." et jusqu'à :"
          INPUT DROP              ( Jusqu'à combien ?)
```

```
1 + SWAP        ( Pour remettre les choses dans l'ordre FORTH)
CLS             ( ou autre présentation, voir WINDOW § 5)
DO
    I DUP DUP
    CR . ." a pour carré"
    U* UD.      ( Pour pouvoir jouer avec des grands nombres)
LOOP ;
```

Notez bien les DROP après les INPUT, et l'utilisation de U* et de UD. ... Il n'est pas nécessaire d'afficher les carrés des nombres négatifs (Le carré d'un nombre est le même que le carré de son opposé), nous pouvons donc en profiter pour étendre le domaine d'utilisation de notre mot, en considérant des nombres non signés.

Chapitre 9

Chaînes de caractères



Une chaîne de caractères, c'est un mot, une phrase, bref une suite composée de caractères quelconques. Elle peut comporter des lettres de l'alphabet, mais aussi des chiffres, ces caractères de contrôle, des espaces, ou d'autres symboles, même des caractères graphiques.
En bref, une chaîne de caractères est une suite de caractères codables avec le code ASCII étendu.

9-1 Représentation des chaînes de caractères

Dans l'ordinateur, les chaînes de caractères sont tout simplement représentées par la suite des codes ASCII des caractères qui les composent. Pour pouvoir les utiliser (modifier, afficher,...), il suffit donc de savoir où sont ces codes. La méthode utilisée est la suivante :
Les mots qui manipulent une chaîne de caractères laissent sur la pile l'adresse du début de la chaîne, et sa longueur (au maximum 255 caractères).

Exemple :
Le mot " (un simple guillement) sert à définir une chaîne de caractères du clavier, en mode direct. Il range dans l'ordinateur, à un endroit convenu, les caractères que vous entrez ensuite, jusqu'à ce que vous tapiez un deuxième guillemet. Nous appellerons **chaînes temporaires** les chaînes de caractères définies de cette façon.

Attention, le premier guillemet est un mot FORTH, il doit être précédé et suivi d'un espace. Le deuxième n'est qu'un terminateur, et n'a pas à être précédé d'un espace (si vous le faites précéder d'un espace, cet espace fera partie de la chaîne... Le fonctionnement est identique à celui du guillemet qui suit le mot ."", que nous connaissons déjà). Par contre, il doit être suivi d'un espace, pour l'isoler du mot suivant.

Notons aussi qu'une chaîne temporaire doit comporter au moins un caractère. Elle ne peut pas être vide.
" COUCOU" . . 6 23432 OK

6 est la longueur du mot.

23432 est l'adresse du premier caractère.

Il est probable que vous obteniez une adresse différente, peu importe (Cela dépend des mots que vous avez définis depuis le démarrage de votre appareil).

Un petit essai pour vérifier :
23432 C@ . 67 OK

Le mot C@ (adr --) dépose sur la pile la valeur de l'octet situé à l'adresse adr, de la même manière que le mot @ (adr --) laisse sur la pile le nombre rangé à l'adresse adr.

67 est bien le code ASCII de la lettre « C »...
23433 C@ EMIT O OK

La deuxième lettre est bien un « O »...

Pour afficher une chaîne de caractères, le mot de base est :
TYPE (adr n --)
Affiche la chaîne de n caractères qui commence à l'adresse adr.

Essayons :
23432 6 TYPE COUCOU OK
23432 4 TYPE COUC OK
23434 3 TYPE UCO OK

Compris ? Ce n'est pas bien compliqué, dans le fond... Rassurez-vous, FORTH dispose d'une grande richesse de mots concernant les chaînes de caractères, et vous n'aurez pratiquement jamais à travailler au niveau des octets.
Mais pour bien comprendre le fonctionnement de ces mots, il est nécessaire de savoir comment FORTH manipule ses chaînes.

Notation conventionnelle
Puisque les mots utilisent souvent la paire de nombres (adresse, longueur), nous la désignerons dans les notations de pile par un seul signe. La notation utilisée en général est « str ». (En anglais, chaîne de caractères se dit « string ».)
Nous écrirons par exemple : **TYPE** (str —) pour dire que TYPE interprétera comme une adresse et une longueur les deux paramètres du dessus de la pile.
Nous parlerons aussi de (str) pour désigner la chaîne de caractères dont les paramètres sont str (adresse et longueur).

Note :
Les mots utilisant les caractères (KEY, EMIT, ...) utilisent comme paramètre (ou laissent sur la pile) le code du caractère. Les mots qui travaillent avec les chaînes de caractères utilisent deux paramètres (adresse et longueur). Il convient donc de ne pas les confondre.

9-2 Variables chaînes de caractères

L'inconvénient de la méthode précédente pour définir une chaîne de caractères est que toutes les chaînes ainsi créées seront rangées au même endroit. Essayez par exemple :
" Chaîne UN" " Chaîne DEUX" .S

Vous pouvez vérifier que les deux chaînes (Chaîne UN) et (Chaîne DEUX) sont rangées à la même adresse... Si vous tapez DDROP pour enlever les paramètres de (Chaîne DEUX), suivi de TYPE (avec les paramètres de (Chaîne UN) sur la pile), vous obtiendrez :
DDROP TYPE Chaîne DE OK (longueur 8)

9-2-1 Définir une variable chaîne de caractères

Si vous comptez utiliser plusieurs fois une chaîne de caractères, il est préférable de la définir comme variable chaîne de caractères, par l'intermédiaire du mot **STRING**. Comme les mots VARIABLE ou CONSTANT, STRING est un mot de définition (qui sert à définir d'autres mots). Il s'utilise sous la forme :
n STRING <nom>
pour définir au dictionnaire un mot qu'on pourra appeler par le nom <nom>, et qui permettra de ranger une chaîne de caractères comportant au maximum n caractères.
Comme pour les variables numériques, on pourra alors accéder au contenu de la chaîne <nom>, pour y ranger une chaîne de caractères, afficher la chaîne

ou la modifier, en écrivant seulement le mot <nom>. Lorsque FORTH lira ce mot, il saura qu'il doit déposer sur la pile les paramètres (adresse et longueur) de la zone où sont rangés les caractères qui le composent.

Exemple :

```
10 STRING MOT
MOT .S          Vous donne sur la pile 2 paramètres :
                — L'adresse où est rangée la chaîne de nom MOT.
                — Sa longueur, qui doit ici être 0.
```

Lorsqu'on crée ainsi une variable, elle est « initialisée » comme chaîne « vide », c'est-à-dire ne comportant aucun caractère, ou encore de longueur 0.

Si vous avez seulement besoin de la longueur d'une chaîne, vous pouvez utiliser le mot :

LEN (str --- n)
donne la longueur de la chaîne (str)

Exemple :

```
" Il fait beau" LEN . 12  OK
MOT LEN . 0  OK
```

De la même façon, le mot **MLEN** donne la longueur maximum de la chaîne que peut contenir une variable chaîne :

MLEN (str --- n)
donne la longueur maximum de la chaîne (str)

```
MOT MLEN . 10  OK
```

MLEN n'a de sens que dans le cas d'une variable chaîne de caractères (définie par le mot STRING).
Une chaîne temporaire n'a pas de longueur maximum. Le nombre déposé sur la pile par MLEN n'a donc pas de signification pour de telles chaînes.

9-2-2 Affectation et affichage des variables chaînes de caractères

Le mot qui affiche une chaîne de caractères, vous le connaissez déjà, c'est TYPE (voir précédemment).
Le mot à utiliser pour ranger une chaîne de caractères dans une variable est $!. Le symbole $ rappelle que le résultat est une chaîne de caractères (comme en BASIC), et le symbole ! rappelle qu'il s'agit d'une affectation, comme pour les nombres.

$! (str1 str2 ---)
Range la chaîne (str1) dans (str2)

Exemple :

```
10 STRING MOT       " SALUT" MOT $!
MOT LEN . 5  OK
MOT TYPE SALUT  OK
```

```
MOT MLEN . 1Ø  OK
" SALUT " MOT $!                    L'espace après le T de SALUT fait
MOT LEN . 6  OK                      partie de la chaîne de caractères...
MOT TYPE SALUT  OK
" Essayons une longue phrase " MOT $!
MOT TYPE Essayons u  OK
MOT LEN . 1Ø  OK
```

Le mot $! a automatiquement tronqué notre longue phrase à 1Ø caractères (longueur maximale d'une chaîne rangée dans la variable MOT).

```
8 STRING SUJET          (groupe sujet)
6 STRING VERBE          (groupe verbal)
8 STRING OBJET          (groupe objet)
: PHRASE       (---)          SUJET TYPE
                              SPACE VERBE TYPE
                              SPACE OBJET TYPE ;
```

Ou bien sûr : PHRASE OBJET VERBE SUJET TYPE SPACE TYPE SPACE TYPE ;

Puis :

```
" aime"  VERBE $!
" Joan"  SUJET $!
" Jill"   OBJET $!          PHRASE   Joan aime Jill  OK
 OBJET  SUJET $!            PHRASE    Jill aime Jill  OK
" Joan" OBJET  $!           PHRASE   Jill aime Joan  OK   Etc.
```

Nous avons déjà signalé qu'une chaîne temporaire ne peut être vide (de longueur Ø). Il est donc impossible d'utiliser le mot guillemet pour réinitialiser une variable chaîne de caractères. Vous pouvez utiliser à cet effet la variable "" (deux guillemets accolés, sans espace entre les deux), qui figure dans le dictionnaire de base.

Attention, "" est un mot FORTH !!!. Il n'a rien à voir avec le mot « guillemet », c'est une chaîne de caractères constante, mais qui se manipule exactement comme une autre variable chaîne de caractères.

```
"" MOT $!     ("" est la « chaîne vide », de longueur Ø)
MOT TYPE  OK        MOT LEN . Ø  OK
```

Si vous n'aimez pas "", vous pouvez utiliser une autre variable chaîne non encore affectée, puisque toute nouvelle variable chaîne créée a pour valeur initiale la chaîne vide.

9-3 Opérations sur les chaînes de caractères

Trois sortes d'opérations peuvent être envisagées :
— Extraction de sous-chaînes dans une chaîne.
— Concaténation (mise bout à bout de deux chaînes).
— Modification d'une partie d'une chaîne.

9-3-1 Extraction de sous-chaînes

On peut extraire la partie gauche (x caractères) d'une chaîne, la partie droite, ou des caractères consécutifs quelconques d'une chaîne, grâce aux mots :

LEFT$ (str1 n --- str2)
RIGHT$ (str1 n --- str2)
MID$ (str1 n1 n2 --- str2)

Ces mots ne créent pas de nouvelles chaînes, mais se contentent de modifier les paramètres str1 de la pile des données. Les paramètres obtenus sont ceux d'une chaîne (str2) définie comme suit :

Pour LEFT$: (str2) est composée des n caractères de gauche de (str1).
Pour RIGHT$: (str2) est composée des n caractères de droite de (str1).
Pour MID$: (str2) est composée des n2 caractères de (str1), à partir du caractère numéro n1. Le premier caractère d'une chaîne a le numéro 1...

Exemples :

```
30 STRING CLAPCLAP
" Le FORTH, c'est chouette" CLAPCLAP $!
CLAPCLAP 7 LEFT$ TYPE          Le FORT  OK
                               ( 7 caractères de gauche)
CLAPCLAP 15 RIGHT$ TYPE        c'est chouette  OK
                               ( 15 caractères de droite)
CLAPCLAP 4 5 MID$ TYPE         FORTH  OK
                               ( 5 caractères à partir du quatrième)
```

Un autre exemple :

Jeu de mots...

```
30 STRING MOT
: JEU          (---)
    MOT DUP                ( duplique la longueur du MOT)
    1+ 1                   ( paramètres de boucle)
    DO
        CR DDUP            ( duplique les paramètres de MOT)
        1 LEFT$ TYPE       ( Affiche les 1 caractères de gauche)
    LOOP
    DDROP ;                ( Pour nettoyer la pile)
```

Essayons :

```
" TRUC" MOT $! JEU
T
TR
TRU
TRUC
```

Exercice :

9.a - En vous inspirant de l'exemple précédent, laissez libre cours à votre imagination... Par exemple, essayez de construire des mots qui à partir du mot « FORTH » puissent afficher :

FORTH		R		F
ORTHF		ORT		FOR
RTHFO		FORTH	FORTH	
THFOR	ou	ORT	ou encore	RTH
HFORT	R	H		

9-3-2 Concaténation de chaînes de caractères

Il s'agit ici de prendre deux chaînes de caractères, et d'en fabriquer une seule en les mettant bout à bout. Cela se fait par le mot $+

$+ (str1 str2 ---)
« Ajoute » la chaîne (str1) à la fin de (str2).
Le résultat est dans (str2).

Exemple :

```
20 STRING OBJET        5 STRING SUJET        5 STRING VERBE
" Jill" OBJET $!       " Joan" SUJET $!      " aime" VERBE $!
```

Puis :

```
VERBE OBJET $+         OBJET TYPE Jill aime  OK
SUJET OBJET $+         OBJET TYPE Jill aime Joan  OK
```

Nous voulons maintenant ajouter "que" devant cette affirmation... Mais attention, on ne peut concaténer qu'à droite. Autrement dit, il est impossible d'ajouter "que" avant "Jill aime Joan". Il nous faut donc ajouter "Jill aime Joan" à la fin de "que"...

```
20 STRING OBJ " que" OBJ $! OBJET OBJ $+
```

Le groupe OBJET est bien maintenant "Jill aime Joan", et la proposition OBJ est "que Jill aime Joan"...

De même :

```
30 STRING PHR
SUJET PHR $!        VERBE PHR $+        OBJ PHR $+
PHR TYPE Joan aime que Jill aime Joan  OK
40 STRING PHR1
" Jill" PHR1 $! VERBE PHR1 $+ " que" PHR1 $+ PHR PHR1 $+
PHR1 TYPE Jill aime que Joan aime que Jill aime Jo  OK
```

Tiens, un problème... PHR1 a une longueur maximum de 40, et ne peut donc contenir une chaîne de longueur 42 (43 avec l'espace de la fin).

```
43 STRING PHR2      PHR1 PHR2 $!      " an" PHR2 $+
PHR2 TYPE      Jill aime que Joan aime que Jill aime Joan  OK
```

Ça va ?? Pas trop perdus ?

9-3-3 Modification d'une chaîne

SUB$! permet de remplacer une partie d'une chaîne par une autre. Sa syntaxe est :

SUB$! (str1 str2 ---)
Remplace (str2) par (str1).
A vous de calculer les paramètres de (str2)...

Exemple :
PHR contient toujours « Joan aime que Jill aime Joan »
Remplaçons le premier « aime » par « veut ».
Il nous faut pour cela remplacer les paramètres de PHR par ceux de la sous-chaîne correspondante.
On peut bien sûr procéder « à la main », en ajoutant 5 à l'adresse de PHR (pour se placer sur le sixième caractère) et en fixant la longueur à 4. Cela donne :

```
" veut" PHR DROP ( enlève la longueur de PHR de la pile) 5 + 4 SUB$!
PHR TYPE Joan veut que Jill aime Joan  OK
```

Mais il est bien plus simple (et plus sûr) de laisser les commandes LEFT$, RIGHT$, ou MID$ faire ce travail à notre place !!!
Le résultat précédent s'obtient plus simplement par :

```
" veut" PHR 6 4 MID$ SUB$!      PHR TYPE ...
```

Il sera presque toujours plus intéressant d'utiliser LEFT$, RIGHT$, ou MID$ que de manipuler les adresses et les longueurs des chaînes. La commande INSTR (voir plus loin) peut souvent aider à trouver la chaîne à remplacer.

Mais tout d'abord, quelques précisions sur le fonctionnement de SUB$! :

```
" voudrait" PHR 6 4 MID$ SUB$!      PHR TYPE      Joan voud que Jill aime Joan
OK
```

Nous n'avons demandé que le remplacement de 4 caractères, pas 8...
De même :

```
" aimerait bien" PHR 6 4 MID$ SUB$!
PHR TYPE      Joan aime que Jill aime Joan  OK
```

En fait, l'action de SUB$! se limitera automatiquement non seulement au nombre de caractères annoncé, mais aussi, si vous remplacez la fin d'une chaîne, au nombre de caractères de la chaîne que vous voulez modifier.

```
PHR LEN . 29  OK      PHR MLEN 3Ø  OK
" Juliette" PHR 25 8 MID$ SUB$! PHR TYPE Joan aime que Jill aime Julie  OK
PHR LEN . 29  OK      (La longueur de PHR a été respectée)
```

Il serait d'ailleurs plus clair d'écrire : " Juliette" PHR 5 RIGHT$ SUB$!
(La limitation agirait ici sur la chaîne " Juliette", au lieu d'agir sur PHR).

Une autre commande bien utile pour trouver la position d'une sous-chaîne est le mot INSTR. Son fonctionnement est le suivant :

INSTR
(str1 str2 --- n)
n est la position du début de la première occurrence de (str1) dans (str2), Ø si (str1) n'est pas trouvée.

Exemple :

PHR contient encore « Joan aime que Jill aime Joan »

```
" aime" PHR INSTR .  6  OK      (recherche de « a-i-m-e »)
" n"    PHR INSTR .  4  OK      (recherche de « n-espace »)
" q"    PHR INSTR . 11  OK      (recherche de « q »)
PHR     PHR INSTR .  1  OK      (normal, non ?)
" bof"  PHR INSTR .  0  OK      (Pas de « bof » dans la chaîne)
```

Faisons par exemple un mot REMPLACE, qui mette une chaîne X dans une phrase Y, à la place d'une chaîne Z... Sur la pile, nous avons les six nombres composant les paramètres de X, Y, et Z.

— Première étape : chercher la position de Z dans Y

DOVER INSTR La pile contient : X Y n --- (n = position de Z dans Y)

— Deuxième étape : si trouvé, mettre sur la pile X et Y1 (chaîne à remplacer)
sinon, restaurer la pile et prévenir (par BELL, par exemple)

Si trouvé : Il faut calculer les paramètres de Y1.

Pour utiliser MID$, la pile doit contenir Y n p --- (p = longueur de X)
DOVER DROP ramène sur la pile la longueur de X. Etat de la pile : X Y n p ---

Après MID$, la pile contient (X Y1 ---), et est prête pour le mot SUB$!.

Si pas trouvé (si n = 0), dépiler 5 nombres (2 chaînes + 1 nombre).

D'où le mot :

```
: REMPLACE (X Y Z --- met X dans Y, à la place de Z)
    DOVER INSTR DUP        (duplique pour comparaison)
        IF DOVER DROP MID$ SUB$!        (Si trouvé, remplacer)
        ELSE DROP DDROP DDROP BELL      (Sinon, dépiler X, Y, n)
        THEN ;
```

Puis : Supposons que PHR contient encore « Joan aime que Jill aime Joan » (voir 9-3-2), et que SUJET contient « Joan ». Nous pouvons taper :

```
" Anne" PHR SUJET REMPLACE PHR TYPE  Anne aime que Jill aime Joan  OK
" Anne" PHR SUJET REMPLACE PHR TYPE  Anne aime que Jill aime Anne  OK
" Anne" PHR SUJET REMPLACE (Fait un bip, il n'y a plus de Joan dans PHR).
```

Mais attention à l'exemple suivant !!!...

```
" Jean" PHR " Anne" REMPLACE PHR TYPE Anne aime que Jill aime Anne  OK
```

Explication : " Jean" et " Anne" sont des chaînes temporaires, de sorte que la chaîne " Anne" est venue se mettre en mémoire à la place de la chaîne " Jean" (Voir exemple au début du 9-2). REMPLACE a donc remplacé " Anne" par " Anne"...

Exemple :
PHR contient encore « Joan aime que Jill aime Joan »

```
" aime" PHR INSTR .  6  OK      (recherche de « a-i-m-e »)
" n"    PHR INSTR .  4  OK      (recherche de « n-espace »)
" q"    PHR INSTR . 11  OK      (recherche de « q »)
PHR     PHR INSTR .  1  OK      (normal, non ?)
" bof"  PHR INSTR .  Ø  OK      (Pas de « bof » dans la chaîne)
```

Faisons par exemple un mot REMPLACE, qui mette une chaîne X dans une phrase Y, à la place d'une chaîne Z... Sur la pile, nous avons les six nombres composant les paramètres de X, Y, et Z.
— Première étape : chercher la position de Z dans Y
DOVER INSTR La pile contient : X Y n --- (n = position de Z dans Y)

— Deuxième étape : si trouvé, mettre sur la pile X et Y1 (chaîne à remplacer)
sinon, restaurer la pile et prévenir (par BELL, par exemple)

Si trouvé : Il faut calculer les paramètres de Y1.

Pour utiliser MID$, la pile doit contenir Y n p --- (p = longueur de X)
DOVER DROP ramène sur la pile la longueur de X. Etat de la pile : X Y n p ---

Après MID$, la pile contient (X Y1 ---), et est prête pour le mot SUB$!.

Si pas trouvé (si n = Ø), dépiler 5 nombres (2 chaînes + 1 nombre).

D'où le mot :

```
: REMPLACE (X Y Z --- met X dans Y, à la place de Z)
    DOVER INSTR DUP       (duplique pour comparaison)
        IF DOVER DROP MID$ SUB$!        (Si trouvé, remplacer)
        ELSE DROP DDROP DDROP BELL      (Sinon, dépiler X, Y, n)
        THEN ;
```

Puis : Supposons que PHR contient encore « Joan aime que Jill aime Joan » (voir 9-3-2), et que SUJET contient « Joan ». Nous pouvons taper :

```
" Anne" PHR SUJET REMPLACE PHR TYPE  Anne aime que Jill aime Joan  OK
" Anne" PHR SUJET REMPLACE PHR TYPE  Anne aime que Jill aime Anne  OK
" Anne" PHR SUJET REMPLACE (Fait un bip, il n'y a plus de Joan dans PHR).
```

Mais attention à l'exemple suivant !!!...

```
" Jean" PHR " Anne" REMPLACE PHR TYPE Anne aime que Jill aime Anne  OK
```

Explication : " Jean" et " Anne" sont des chaînes temporaires, de sorte que la chaîne " Anne" est venue se mettre en mémoire à la place de la chaîne " Jean" (Voir exemple au début du 9-2). REMPLACE a donc remplacé " Anne" par " Anne"...

9-4 Comparaison de chaînes de caractères

On peut aussi comparer des chaînes de caractères. L'ordinateur vous dira si un mot est inférieur, supérieur ou égal à un autre...
Qu'est-ce que cela signifie ?... Eh bien, cela se passe exactement comme lorsque vous cherchez dans un dictionnaire. Si un mot est avant un autre, nous dirons qu'il est inférieur. S'il est après, il sera supérieur...
A cela près que votre ordinateur compare tous les caractères du code ASCII, y compris les espaces et les caractères de contrôle, au lieu des lettres de l'alphabet.

Les deux mots qui servent à comparer les chaînes de caractères sont les suivants :

$= (str1 str 2 --- cond)
laisse sur la pile 1 si (str1)=(str2), Ø sinon

$< (str1 str2 --- cond)
laisse sur la pile 1 si (str1)<(str2), Ø sinon

Exemples :

```
10 STRING AA        8 STRING BB
"" AA $= . 1  OK
AA "" $= . 1  OK        (quelle chance !!)
" ABC" AA $!      AA BB $!
AA BB $= . 1  OK        (prévisible, non ?)
" ABC " BB $!
AA BB $= . Ø  OK        (il y a un espace à la fin de (BB))
                        (Voir au Manuel de Référence le mot -TRAILING)
                        (pour supprimer les espaces en fin de mot...)
AA BB $< . 1  OK        (Normal, (BB) est plus long que (AA)...)
" AB1" AA $< . 1  OK    (Le code ASCII de "1" est 49 et celui de)
                        ("C" est 67. "AB1" est donc avant "ABC".)
"" AA $<. 1  OK
```

Plus gênant sera peut-être :

```
" abc" BB $!            (AA contient encore "ABC")
AA BB $ = . Ø  OK       (Pas assez malin pour ça)
```

En effet, les codes des caractères minuscules sont supérieurs aux codes des caractères majuscules.

Un piège grossier

Essayons pour voir la commande suivante :

```
" BLANC" " NOIR" $= . Ø  OK
" BLANC" " ROUGE" $= . 1  OK (Que lui arrive-t-il ?...)
```

Non, il ne s'est pas trompé. Souvenez-vous que les mots définis par le mot guillemet sont tous rangés au même endroit. Voici ce qui s'est passé :
" BLANC" a rangé le mot (BLANC) à l'endroit convenu, et a laissé sur la pile cette adresse, et la longueur du mot (5). Puis, " ROUGE" a fait de même. $= est alors allé comparer les 5 caractères à cette adresse (avec eux-mêmes !!!). Il y a trouvé deux fois les codes de R, O, U, G, E (Lors de l'exécution de la commande " ROUGE", les caractères B, L, A, N, C ont été remplacés).

De la même manière, le résultat précédent était un peu chanceux, car l'ordinateur a en fait comparé les chaînes " NOIR" et " NOIR" (au lieu de " BLANC")...

Moralité :
Utilisez dans la mesure du possible des variables, et ne laissez pas vos chaînes de caractères traîner dans les zones temporaires... Ajoutons que certains mots du FORTH utilisent aussi ces zones...

Exercices :

9.b - Fabriquer les mots $> et $> =
9.c - Faire un mot REMPLACE-PARTOUT (X Y Z ---) qui remplace dans la chaîne de caractères (Y) toutes les occurrences de (Z) par (X).

9-5 Tableaux de chaînes de caractères

Les chaînes de caractères aussi peuvent se mettre en tableaux. La seule restriction est que toutes les chaînes d'un même tableau doivent avoir la même longueur maximum.
Pour définir un tableau nommé ARTICLES et pouvant contenir 5 chaînes comportant chacune un maximum de 15 caractères, nous écrirons :

```
15 5 ARRAY$ ARTICLES
```

Les éléments du tableau seront numérotés de Ø à 4, comme pour les tableaux de nombres. Pour accéder au troisième élément du tableau, il suffira de spécifier, comme pour les tableaux numériques :

```
2 ARTICLES
```

Mais ici, c'est l'adresse et la longueur de la troisième chaîne du tableau qui arriveront sur la pile, prêtes à être utilisées comme les paramètres d'une variable chaîne de caractères.

Exemple :
Fabriquons un mot BOITES – RESERVE – ? qui affiche l'état de la réserve de l'épicier du chapitre 3 (Voir exercice 3.c).

```
10 5 ARRAY$ NOMS-CONSERVES
" Choucroute"   Ø NOMS-CONSERVES $!
" Cassoulet"    1 NOMS-CONSERVES $!
" Couscous"     2 NOMS-CONSERVES $!
" Potée"        3 NOMS-CONSERVES $!
" Raviolis"     4 NOMS-CONSERVES $!
```

Le tableau RESERVE-CONSERVES (§ 3-4) contient les quantités de boîtes en réserve.

```
: BOITES-RESERVE-?        (---)
        10 1 LOCATE . " BOITES EN RESERVE"
        50 18 269 18 LINE
        5  0  DO          (pour chacune des rubriques...)
              7 1 3 + LOCATE
              I NOMS-CONSERVES TYPE        (affiche le nom)
              20 I 3 + LOCATE
              I RESERVE-CONSERVE ?         (affiche l'état de la réserve)
              LOOP ;
```

Nous pouvons améliorer notablement la présentation à l'écran en « justifiant » les nombres à droite, grâce au mot .R (n p ---), qui affiche le nombre n cadré à droite dans une zone de p espaces.
Il vous suffit pour cela de remplacer le point d'interrogation (dans « RESERVE-CONSERVES ? ») par les mots "@ 5 .R"
L'avant dernière ligne devient :

```
I RESERVE-CONSERVE @ 5 .R
```

Essayons :

```
BOITES-RESERVE-?        Et voilà le travail !
```

9-6 Poser des questions en cours d'exécution

Il arrive souvent qu'on ait besoin en cours d'exécution de poser des questions à l'utilisateur, et d'attendre une réponse de sa part.
La conception même de FORTH rend ce genre de situation moins fréquent que dans les autres langages (puisqu'on tape souvent directement au clavier des séquences de commandes, au lieu d'exécuter un programme figé comme dans la plupart des autres langages), mais cela vous arrivera certainement.
Ne fut-ce que lorsque vous aurez écrit une application qui nécessite plusieurs paramètres, et que vous voudrez qu'elle soit utilisable par d'autres, sans que vous ayez à fournir un mode d'emploi détaillé...

Nous disposons à cet effet de deux mots voisins, **INPUT$** et **LINPUT$**...

INPUT$ (--- str)

Attend qu'on tape une chaîne de caractères au clavier, et la range en zone temporaire, exactement comme si on l'avait entrée par le mot guillemet (vous aurez donc intérêt à mettre au plus tôt cette chaîne dans une variable).
La chaîne ne peut excéder 80 caractères, et est terminée lorsque l'utilisateur tape la touche ENTRÉE.
Après la frappe de la touche ENTRÉE, FORTH retourne à ses occupations en cours, après avoir déposé sur la pile les paramètres de la zone où vous trouverez la chaîne tapée au clavier.

votre répertoire, supprimer des noms, etc. Mais même ici c'est plus « FORTH ».

```
: DEMANDE        ( str1 n1 n2 -- str2)
        LOCATE TYPE INPUT$ ;
: DEMANDER-NOM      ( -- str)
        " NOM :" 0 10 DEMANDE ;
: DEMANDER-ADRESSE      ( -- str)
        " ADRESSE :" 0 12 DEMANDE ;
```

D'une manière générale, n'ayez pas peur d'écrire des mots courts. Ils seront plus faciles à comprendre et à tester. En outre, vous pourrez souvent leur donner des noms bien significatifs, alors que les « gros mots » réalisent souvent des actions complexes, agissent sur des paramètres nombreux, et sont plus délicats à tester, à appréhender, à corriger...

Voyez l'annexe relative aux périphériques, comment sauvegarder (ou relire) votre tableau sur une cassette magnétophone. Un moyen est de considérer un buffer comme un tableau particulier... (voir l'exemple « Annuaire Téléphonique »).

Annexe A

L'éditeur

Les mots FORTH qu'on définit au clavier prennent immédiatement place dans le dictionnaire, et il n'est plus possible de modifier le texte qui a servi à les créer. Tout au plus peut-on oublier la nouvelle définition (FORGET), et tout recommencer.

En effet les mots que vous avez tapés (on parle en général du « code source », ou de « texte ») sont utilisés par FORTH pour fabriquer un « code objet » (ou « code FORTH »), la matière première qui constitue le dictionnaire.
Ce processus est en principe irréversible, de sorte que le seul moyen de conserver le code-source est de ne pas le mettre au dictionnaire immédiatement, mais de le conserver dans une zone où FORTH ne peut pas aller écrire inopinément (Sauf bien sûr si nous lui demandons, on peut TOUT faire, en FORTH...). C'est exactement ce que fait l'éditeur. Mais en outre, il nous donne des facilités d'écriture, de modification, et de sauvegarde sur cassette (ou sur disquette) des textes.

A-1 Principes généraux

Lorsque vous êtes sous le contrôle de l'éditeur, le défilement de l'image est interrompu, et l'écran de votre téléviseur est recopié fidèlement dans une zone réservée, qu'on appelle traditionnellement une « zone tampon », ou en anglais un « buffer ».
L'écran comporte 25 lignes de 40 caractères, donc un buffer comporte de la place pour conserver 1000 caractères (en réalité 1024).
Il y a plusieurs buffers (en principe 4), de sorte qu'on peut conserver l'image de plusieurs écrans du téléviseur. On utilisera aussi le mot « écran » (en anglais SCREEN, abrégé en SCR) pour désigner l'image-buffer d'un écran du téléviseur.

Nous disposerons alors de 2 groupes de commandes :
— Des commandes d'édition proprement dite, c'est-à-dire de modification des écrans, et de sortie de l'éditeur. Cela se fait simplement en modifiant votre écran de télévision, puisque le buffer est à chaque instant une reproduction exacte de cet écran.
— Des commandes de communication avec les périphériques. On peut écrire les écrans sur cassette, sur disquette, ou sur imprimante. Et bien sûr on peut aussi les relire...

Puisqu'on peut sauvegarder les écrans sur cassette, il faut leur donner un nom (sinon, comment les rappeler ?)... En fait, on leur donne un numéro, qu'on peut choisir librement entre 1 et 999 (surtout pas Ø).
Pour éditer l'écran numéro 1Ø, la commande est :
1Ø EDIT
On ne peut bien sûr éditer qu'un écran à la fois, mais sauf indication contraire, on dispose de 4 buffers, de sorte qu'on peut garder en mémoire le contenu de 4 écrans.
Par ailleurs, on peut modifier le nombre de buffers, et aller jusqu'à 20 buffers en version cassette !!!. Mais dans ce cas extrême, le dictionnaire n'aurait plus de place pour s'étendre, et nous ne pourrions pas y « compiler » les écrans. Cette utilisation est toutefois imaginable pour par exemple utiliser l'éditeur pour ranger (ou traiter) des données, ou encore comme mini traitement de texte.

A-2 Commandes d'édition

A-2-1 Commandes « ordinaires »

Vous avez tapé :
1Ø EDIT

Votre écran est vierge, et le curseur est en haut à gauche.
Tapez par exemple une ligne de « A », en maintenant le doigt appuyé sur la touche, même après la fin de ligne... Quel gargouillis !!!

Arrivés en bout de ligne, nous ne passons pas sur la ligne suivante. Cela est volontaire, c'est pour faciliter une « bonne » présentation des programmes. Une ligne représentera souvent une entité logique dans les définitions, il est normal que l'affichage ne change pas de ligne sans nous demander notre avis. Pour changer de ligne, il faut le faire exprès.

Pour donner des ordres à l'éditeur, on utilise les touches « spéciales » du clavier (ENTRÉE, les flèches, STOP, RAZ, INS, EFF, et CNT)

ENTRÉE a pour seul effet de placer le curseur au début de la ligne suivante.
Tapons la touche ENTRÉE. Théoriquement, nous devrions obtenir le message « UK ERROR »... Eh bien non. Tant que nous n'en aurons pas donné l'ordre, tout ce que nous taperons sera mis en réserve sans que FORTH essaie de comprendre...
L'éditeur sert en quelque sorte de paravent.
Vous pouvez aussi essayer les flèches : on peut librement déplacer le curseur dans tout l'écran. La flèche ← place le curseur alternativement en début de première et de dernière ligne.
Noter aussi que le curseur s'arrête aux bords de l'écran. Cela est conforme à la logique de l'éditeur.

RAZ (Remise à Zéro) efface tout, comme son nom l'indique.
Tapons RAZ pour effacer notre vilaine ligne de « A », et écrivons un texte

plus « significatif ». Par exemple : plaçons-nous en ligne 12, colonne 7, et tapons :

DANS QUELQUES INSTANTS
LA SUITE (ligne 13)
DE NOTRE EMISSION (ligne 14)

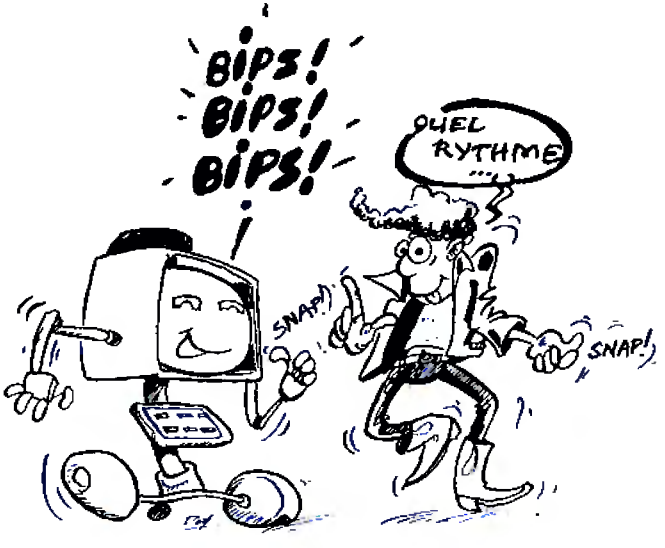
STOP arrête l'édition, et retourne sous FORTH sans autre forme de procès. Vous voyez d'ailleurs réapparaître le OK qui vous rappelle que vous êtes en mode « interactif », et que ce que vous allez taper sera immédiatement exécuté.

Toujours pas de message UK ERROR (à moins que vous n'ayez appuyé deux fois sur la touche STOP). La commande STOP ne transmet pas au FORTH le texte que vous venez de taper, elle sert seulement à quitter l'éditeur. Cela peut aussi être utile lorsqu'on veut utiliser l'éditeur pour autre chose que pour écrire des mots FORTH (pour écrire directement dans des fichiers, par exemple...).

Car votre écran n'est pas perdu !!. Si vous retapez 1Ø EDIT, vous le retrouverez dans l'état où vous l'aviez laissé (L'éditeur vient simplement de recopier le buffer à l'écran). Si par hasard vous êtes sorti de l'éditeur et que vous avez oublié le numéro de l'écran concerné, REDIT éditera automatiquement le dernier écran utilisé.

EFF supprime le caractère à la position du curseur, et décale la fin de la ligne vers la gauche.

INS insère un espace à la position du curseur, et décale la fin de la ligne vers la droite. Lorsque la ligne est pleine, c'est-à-dire lorsque le fait d'insérer un espace nous fait perdre un caractère en fin de ligne, nous sommes prévenus par un bip. A nous alors de décider si nous voulons continuer à insérer... Comme on le voit, la touche INS peut aussi servir à effacer des mots en fin de ligne... (Et en musique !!!).



Reprenons notre message, et transformons-le.
Avec les flèches, plaçons le curseur au début de « QUELQUES », et tapons « UN », suivi de 6 fois [EFF], pour effacer « ELQUES ».
Quelques coups sur la flèche à droite, puis de nouveau [EFF] pour effacer le « S » à la fin de « INSTANTS ».
Nous allons remplacer maintenant « LA » par « UNE ». Plaçons-nous sur le « L », et tapons « UNE ». Puis insérons un espace avec [INS], pour séparer « UNE » et « SUITE ».
Allons ensuite sur le « N » de « NOTRE », et tapons « BIPS ». Il ne reste qu'à effacer « E EMISSION ». On peut utiliser [EFF], mais il est aussi amusant de l'effacer en fin de ligne, en utilisant [INS] (En tous cas, c'est plus musical...)
Votre écran devrait contenir maintenant :

```
DANS UN INSTANT
      UNE SUITE
   DE BIPS
```

Si vous en êtes à votre premier contact avec l'éditeur, vous en avez déjà assez pour travailler agréablement. La seule chose qui vous manque est la commande normale de sortie :

[CNT]-[Y] qui retourne en FORTH et traite ce que vous avez entré exactement comme si vous l'aviez tapé directement. Les mots tapés entre « : » et « ; » seront mis au dictionnaire, les autres seront exécutés.
Tapons [CNT]-[Y]... DANS ? UK ERROR C'était prévisible, non ???

C'est ici qu'apparaît le plus gros intérêt de l'éditeur, votre écran est toujours là ! Si maintenant vous tapez REDIT (ou 1Ø EDIT), votre écran est revenu, et vous êtes de nouveau en édition, prêts à modifier ou corriger vos mots, ou encore à continuer votre programme.
Essayons par exemple de donner un sens à tout cela...
REDIT
pour rappeler l'écran numéro 1Ø.
Dans les 8 premières lignes de l'écran, tapons fidèlement les 8 lignes suivantes (y compris les parenthèses)

```
: BIPS BELL ;      (hé, hé)
: DE BIPS ;        (bizarre...)
: SUITE DE BIPS ;
: UNE SUITE DE BIPS ;
: INSTANT UNE SUITE DE BIPS ;
: UN INSTANT UNE SUITE DE BIPS ;
: DANS UN INSTANT UNE SUITE DE BIPS ;
DANS UN INSTANT UNE SUITE DE BIPS
```

[CNT]-[Y]
Nous avons bien les bips promis (combien y en a-t-il ?).
FORTH a d'abord mis au dictionnaire les mots définis par « : » et « ; » puis a exécuté la commande « DANS UN INSTANT UNE SUITE DE BIPS » comme si vous l'aviez tapée au clavier.
Vérifications : VLIST pour vérifier la présence des mots, et vous pouvez toujours retaper en mode direct la même commande, vous aurez le même résultat. Pas un bip de plus, pas un de moins.

Souvent, lorsqu'on veut écrire des mots, on utilise l'éditeur pour les définir, et à la fin de l'écran on les essaie comme s'ils provenaient du clavier, avant d'oublier tout l'écran en écrivant à la fin FORGET xxxx (nom du premier mot de l'écran).
Lorsqu'on tapera [CNT]-[Y], les mots seront mis au dictionnaire, exécutés, puis oubliés. On peut même terminer l'écran par REDIT pour retourner automatiquement sous l'éditeur, si l'exécution de notre écran nous laisse le temps d'apprécier les résultats de notre travail.

Exemple :
Tapez :
FORGET BIPS
pour oublier ces mots superflus.
REDIT
pour retourner en édition de l'écran 1Ø.

Puis en dernière ligne de l'écran : (tapez une ou deux fois la flèche en « U »)
FORGET BIPS REDIT
Terminez par
[CNT]-[Y] ... N'est-ce pas fantastique ???
Vérification : Tapez la touche STOP
VLIST... Pas de trace de nos mots...

Une autre méthode plus fréquemment utilisée consiste à insérer :
FORGET XX : XX ;
en première ligne de votre écran (XX ou un autre nom...). Bien sûr, il faudra taper : XX ; avant d'entrer sous l'éditeur, mais cette méthode offre l'avantage de laisser vos mots dans votre dictionnaire, de sorte que vous pouvez les tester de nouveau comme vous l'entendez.

Assez joué, pour un premier contact avec l'éditeur c'est suffisant. Les commandes qui suivent ne sont là que pour votre confort, et ne sont pas indispensables à l'étude du FORTH. Vous pourrez les consulter lorsque vos applications seront d'un volume suffisant pour justifier l'envie de déplacer des lignes ou des écrans « en bloc ».

A-2-2 Commandes « grand confort »

Toutes les autres commandes de l'éditeur s'obtiennent par la frappe simultanée de la touche [CNT] (contrôle) et d'une autre touche.

[CNT]-[A]

[CNT]-[Z] donnent les crochets, comme en mode direct.
(CNT-A crochet ouvrant, CNT-Z crochet fermant)

[CNT]-[D] (Début) place le curseur alternativement en début et fin de la ligne où est le curseur.

[CNT]-[F]

CNT-G provoquent une tabulation jusqu'à la prochaine position tabulée du curseur. Ces positions correspondent aux colonnes 1, 6, 11, 16, 21, 26, 31, 36. (CNT-F tabule vers la gauche, CNT-G tabule vers la droite)

CNT-N (Numéro ?) vous rappelle le numéro de l'écran en cours d'édition, si vous l'aviez oublié. Une deuxième frappe de CNT-N efface le numéro.

CNT-B même effet que la touche STOP
CNT-H même effet que la flèche à gauche
CNT-I même effet que la flèche à droite
CNT-J même effet que la flèche vers le bas
CNT-K même effet que la flèche vers le haut
CNT-L même effet que la touche RAZ (attention aux erreurs de frappe !!)
CNT-M même effet que la touche ENTRÉE

CNT-S (Scinder ou inSérer) scinde en deux la ligne du curseur. La partie de la ligne située à la droite du curseur est déplacée à la ligne suivante, et toutes les lignes qui étaient auparavant en dessous du curseur sont décalées vers le bas.
La partie droite de la ligne du curseur est effacée.
S'il n'y a pas assez de place sous le curseur (dernière ligne écrite), la commande restera sans effet.
Pour insérer une ligne blanche, il suffit de positionner le curseur au début de la ligne (par CNT-D, ENTRÉE et les flèches), et de taper CNT-S.

CNT-U (réUnir) réunit deux lignes consécutives. (inverse de CNT-S)
Les caractères de la ligne suivante sont replacés à droite du curseur, et les lignes qui suivent sont déplacées vers le haut.
Attention : la partie de votre ligne située à droite du curseur sera définitivement perdue.
On peut supprimer une ligne en se plaçant au début, et en tapant CNT-U. Mais à la différence de CNT-X (ci-dessous), la ligne n'est pas mise en réserve.
S'il n'y a pas assez de place à droite du curseur pour mettre la ligne suivante, la commande sera inopérante.

CNT-X (faire une croix) supprime la ligne où est actuellement le curseur. Les lignes suivantes sont remontées d'une position, et la dernière ligne est mise à blanc.
Mais votre ligne supprimée n'est pas perdue définitivement !!!
Elle est mise en réserve, et vous pouvez la rappeler par CNT-E.
Si la réserve est pleine, votre ligne est supprimée mais n'est pas sauvegardée. Dans ce cas, vous entendrez un bip.
CNT-X et CNT-E sont très utiles pour déplacer des lignes, même d'un écran sur un autre, à condition de ne pas quitter l'éditeur.
(Voir les commandes CNT-O et CNT-Q)

CNT-E (Écrire) rappelle la dernière ligne mise en réserve, et l'insère à la ligne où se trouve actuellement le curseur.
L'ancienne ligne du curseur et les suivantes sont décalées vers le bas.

Si aucune ligne n'avait été mise en réserve, pas d'action.
Cette commande est souvent utilisée conjointement à [CNT]-[X] (pour déplacer des lignes), ou à [CNT]-[C] (pour recopier des lignes).
On peut ainsi déplacer ou recopier des écrans entiers, à condition de n'avoir pas choisi trop de buffers.
Les lignes mises en réserve sont empilées (eh oui, encore !!!), de sorte que la dernière mise en réserve est la première rappelée.

[CNT]-[C] (Copier) met en réserve la ligne actuelle du curseur, sans modifier l'écran.
Si la réserve est pleine, votre ordinateur émet un bip, et la commande reste sans effet. (On peut vider la réserve par [CNT]-[T])

[CNT]-[T] (iniTialisation de la réserve Temporaire) vide le buffer où sont mises en réserve les lignes détruires par [CNT]-[X], ou recopiées par [CNT]-[C]. Toutes les lignes en réserve sont définitivement perdues.

[CNT]-[O]
[CNT]-[Q] Si un écran ne vous suffit pas, ces commandes vous mettent en édition sur l'écran précédent ([CNT]-[O]) ou sur l'écran suivant ([CNT]-[Q]).
Si vous étiez dans l'écran 10, [CNT]-[Q] vous permettra d'éditer l'écran 11. L'écran 10 n'est pas perdu (vous pouvez y revenir par [CNT]-[O]), mais si vous aviez 4 buffers et que vous tapez 4 fois [CNT]-[Q], vous le perdrez. Au quatrième [CNT]-[Q], vous éditerez l'écran 14 dans le buffer où était l'écran 10. Ainsi se fait l'allocation des buffers.
...1.2.3.4.1.2.3.4.1.2.3... (Et ainsi de suite...)

A-3 Éditeur et périphériques

Nous ne parlerons ici que des cassettes et de l'imprimante. La gestion des disquettes par l'éditeur est légèrement différente (moins contraignante) et sera traitée séparément.

A-3-1 L'imprimante

[CNT]-[P] (Printer) recopie l'écran en cours sur votre imprimante, en rappelant le numéro en en-tête.
Vérifiez bien que votre imprimante est connectée et en fonctionnement, sinon le système se met en attente et vous n'aurez plus qu'à appuyer sur le bouton INITIALISATION PROGRAMME.
Mais même dans ce cas, rassurez-vous, rien n'est perdu, il suffit de taper REDIT pour retrouver votre écran comme il était.

A-3-2 Les cassettes

CNT-W (Write) Vous êtes content de vous, et vous désirez sauvegarder le texte de votre écran pour le montrer à vos amis ? (ou pour y retravailler demain...). Rien de plus facile, mettez votre cassette en enregistrement (de préférence pas à l'endroit d'un programme auquel vous tenez), et tapez CNT-W

Aussitôt, le moteur du magnétophone se met en marche, et enregistre votre écran.

A la suite de quoi nous nous retrouvons sous l'éditeur...

Notes

— Votre écran a été sauvegardé sous le nom « BLOCK1Ø.DAT ». S'il avait porté le numéro 23, par exemple, il aurait été sauvegardé sous le nom « BLOCK23.DAT ».

Les écrans FORTH sont lisibles depuis le BASIC (comme fichiers à accès direct), de sorte qu'on peut s'amuser à écrire les fichiers ou des programmes BASIC en utilisant l'éditeur du FORTH...

— FORTH refusera de sauvegarder deux fois un même écran si vous ne l'avez pas modifié entre-temps (Il suffit de retaper un caractère quelconque de l'écran pour qu'il soit considéré comme modifié).

Cela n'est pas une erreur, c'est une anticipation sur la gestion des disquettes (en mémoire virtuelle).

CNT-R (Read) relira alors depuis la cassette l'écran que vous êtes en train d'éditer (C'est-à-dire celui qui porte le même numéro).

Il faut évidemment que cet écran ait déjà été sauvegardé, sinon vous allez voir se dérouler votre cassette, sans autre recours que d'appuyer sur le bouton d'initialisation pour arrêter le processus.

Vous venez juste de sauvegarder l'écran numéro 1Ø, faisons l'expérience. Rembobinez votre cassette pour revenir avant l'écran, et vidons le buffer correspondant. Peu importe la méthode :

Depuis l'éditeur, vous pouvez taper la touche RAZ,

Vous pouvez repartir en FORTH (STOP) et taper EMPTY-BUFFERS,

Vous pouvez couper le courant et le rallumer.

Dans les deux derniers cas, il vous faudra revenir sous l'éditeur par 1Ø EDIT (ou par REDIT, si vous n'avez pas coupé le courant)

Tapez CNT-R... Pas de problème, votre écran réapparaît exactement dans l'état où vous l'aviez quitté.

A-4 L'éditeur vu du FORTH

L'éditeur du FORTH est un outil très utile et très puissant pour la mise au point des « programmes ».

Bien sûr, étant incorporé dans FORTH, il travaille sous le contrôle du langage. D'ailleurs, pour ne rien vous cacher, il est écrit entièrement en FORTH. On peut, du FORTH, appeler l'éditeur, ou utiliser les écrans créés sous l'éditeur. Les mots FORTH utiles pour cela sont :

EDIT n ---

Entre dans l'éditeur. Si l'écran numéro n est dans un buffer, l'éditeur va le chercher, et travaille sur cet écran. Sinon, (en version cassette) il met à blanc un buffer (le plus anciennement utilisé) et travaille avec un écran initialement vierge. On peut lire l'écran n sur la cassette en utilisant la commande CNT-R.

(En version disquette, l'éditeur lira automatiquement l'écran sur la disquette s'il n'est pas déjà dans un buffer)

REDIT ---

Retourne en édition sur le dernier écran antérieurement utilisé. Ceci est utile si vous aviez oublié le numéro de l'écran avec lequel vous venez de travailler, ou encore si vous travaillez dans une base inhabituelle (en base 16, il faudra taper A EDIT pour charger l'écran numéro dix (décimal). 1Ø EDIT chargerait l'écran « 16 décimal ».

WHERE ---

Si nous avons tapé CNT-Y pour sortir de l'éditeur, et si FORTH a trouvé une erreur dans notre écran, nous pouvons taper WHERE. Nous retournerons dans l'éditeur, et le curseur sera positionné juste APRÈS le mot qui a causé l'erreur.

Cela est bien utile pour localiser les erreurs lors de la compilation.

On peut aussi utiliser WHERE après LOAD (voir ci-dessous), en cas d'erreur lors de l'interprétation d'un écran ou d'une série d'écrans.

LIST n ---

affiche le contenu de l'écran numéro n. L'affichage se termine par un OK après la dernière ligne, de sorte que le haut de l'écran disparaît. Pour remédier à cet inconvénient, on peut arrêter le défilement, exactement comme pour VLIST (n'importe quelle touche interrompt provisoirement le défilement, puis toute touche autre que STOP le fait continuer ; la touche STOP l'arrête définitivement, et stoppe l'exécution du mot).

Ex :

```
1Ø LIST 11 LIST 12 LIST
ou
PRINTER 1Ø LIST 11 LIST CONSOLE
```

LOAD n ---

interprète l'écran numéro n, exactement comme le ferait CNT-Y sous l'éditeur. Si l'écran n n'existe pas, il sera chargé à partir de la cassette. Cette commande peut surtout être intéressante si vous avez une grosse application. Si par exemple votre application occupe les écrans 1Ø, 11, 12, 15, 18 et 19, il vous suffit de terminer l'écran 1Ø par 11 LOAD, l'écran 11 par 12 LOAD, etc. Lorsque vous taperez ensuite 1Ø LOAD, c'est toute votre application qui sera chargée.

Une autre pratique est de définir un écran « de chargement », par exemple l'écran 1ØØ, qui comporte seulement la suite 1Ø LOAD 11 LOAD 12 LOAD... Vous taperez ensuite 1ØØ LOAD pour tout charger. Dans le cas où votre application est très grosse, cette méthode est la plus simple.

Si votre application occupe des écrans consécutifs, vous pouvez aussi utiliser le mot **THRU** (voir Manuel de Référence).

SCR --- adr
est une variable contenant le numéro du dernier écran utilisé
Le mot REDIT, par exemple, est simplement défini par :
: REDIT SCR @ EDIT ;

Enfin, pour terminer, les écrans de l'éditeur sont placés dans les buffers où sont situés les blocks issus de la cassette ou de la disquette. Tous les mots agissant sur les blocks peuvent donc être utilisés (voir annexe B).
Ces mots ne sont pas liés à l'éditeur, mais agissent sur ses écrans, ou sur le nombre d'écrans qu'on peut avoir simultanément en mémoire (au détriment de la place que pourrait occuper votre dictionnaire).
Entre autres,

EMPTY-BUFFERS met à blanc tous les écrans en mémoire.
SAVE-BUFFERS sauvegarde les écrans modifiés.
MAXBUF (--- n)
donne le nombre maximum d'écrans possibles en mémoire.
BUFFERS (n ---)
permet de fixer le nombre d'écrans utilisables.
#BUF (--- n)
donne le nombre actuel de buffers.
BLOCK (n --- adr)
donne l'adresse du début du buffer numéro n.
(et le charge s'il n'est pas déjà en mémoire).

Exemple :
Lors de l'initialisation de votre système :

```
#BUF . 4  OK
MAXBUF . 22  OK
16 BUFFERS          (Vous travaillez maintenant avec 16 buffers)
#BUF . 16  OK
```

Annexe B

Les périphériques

On appelle « périphériques » tout ce qui vous permet de communiquer avec l'intérieur de votre ordinateur, dans un sens ou dans l'autre.
Le clavier et l'écran sont en fait des périphériques, ainsi que le crayon optique, et les manettes de jeu.
Les autres périphériques courants dont nous n'avons pas encore parlé sont l'imprimante, le lecteur-enregistreur de programmes, et les lecteurs de disquettes. Pour ces derniers, consultez le MANUEL DE RÉFÉRENCE.

B-1 L'imprimante

Si vous disposez d'une imprimante, vous pouvez la « sélectionner en sortie », par le mot :
PRINTER (---)
qui dirige l'affichage vers l'imprimante.

Dorénavant, tout le texte (pas le graphique) qui apparaissait à l'écran sortira sur votre imprimante.
Vous pourrez à tout instant revenir à l'affichage à l'écran grâce au mot :
CONSOLE (---)
qui sélectionne l'écran comme organe de sortie

Donnons quelques exemples :
— Voulez-vous une liste écrite des mots du dictionnaire ?
Tapez simplement :
PRINTER VLIST CONSOLE

Vous pouvez comme auparavant interrompre l'impression en tapant une touche quelconque, suivie de la touche STOP.

— Vous êtes hors de l'éditeur, et vous voulez imprimer le contenu de l'écran 12.
Tapez donc :
PRINTER 12 LIST CONSOLE

LIST a pour effet d'afficher le contenu de l'écran, sans rentrer dans l'éditeur.

— Si vous faites souvent des listings d'écrans sur votre imprimante, il devient intéressant de définir un mot exprès, qui liste les écrans depuis le numéro n1 jusqu'au numéro n2...

Par exemple :

```
: DOC           ( n1 n2 ---)
    PRINTER       ( Imprimante en sortie)
    1 + SWAP DO   ( Prépare la boucle)
       I LIST     ( Liste l'écran numéro I)
    LOOP
CONSOLE ;         ( Retour à l'écran)
```

Utilisation : 10 15 DOC liste les écrans 10 à 15.

Si vous n'avez pas d'imprimante, ou si votre imprimante n'est pas connectée, votre ordinateur vous répondra « ? IO ERROR » lorsqu'il aura à exécuter le mot PRINTER.
Si votre imprimante est connectée mais que rien ne se passe lorsque vous lui demandez d'imprimer quelque chose, vérifiez qu'elle est correctement branchée, et allumée...

Si d'autre part vous avez une imprimante thermique PR90-040, vous pouvez aussi réaliser une (et même plusieurs si vous voulez !) recopies de votre écran graphique, grâce à l'ordre **SCRPRT** (---).
L'écran de votre téléviseur sera reproduit point par point sur papier, texte et graphique.

B-2 Le lecteur-enregistreur de programmes (LEP)

Vous pouvez utiliser votre LEP de 2 manières différentes.
— Pour sauvegarder ou relire des écrans de l'éditeur, ou d'une manière plus générale des données ou du texte.
— Pour sauvegarder l'état actuel de votre dictionnaire, ou le recharger.

B-2-1 Sauvegarder ou relire votre dictionnaire

Êtes-vous satisfait de l'état actuel de votre dictionnaire ?
Si oui, mettez votre magnétophone à cassettes en enregistrement, et tapez par exemple « 1 SAVE ». Votre cassette commencera à tourner, et toute la partie de votre dictionnaire qui ne fait pas partie du FORTH de base (Celle qui apparaît avant le mot « GO » lorsque vous faites un VLIST) sera enregistrée.

SAVE (n ---)
Sauvegarde la partie « utilisateur » du dictionnaire.
(Celle qui n'est pas en cartouche-ROM).
En l'absence de disquettes, il s'agit de la partie du dictionnaire qui est au-dessus du mot GO.

Votre dictionnaire sera enregistré sur cassette sous le nom « DICT1 FTH » (avec 3 blancs). Si vous aviez tapé « 33 SAVE », il aurait porté le nom « DICT33 FTH » (avec 2 blancs).

Pour retrouver votre dictionnaire dans l'état où il était lorsque vous avez tapé SAVE, il vous suffit maintenant de taper « 1 RUN » (ou « 33 RUN » si vous aviez tapé « 33 SAVE »)

RUN (n ---)
Cherche sur la cassette la sauvegarde-dictionnaire qui porte le numéro n, et la transfère dans l'unité centrale.

Si, en outre, vous voulez qu'après avoir chargé le dictionnaire (par RUN) FORTH exécute automatiquement un mot, il vous suffit de lui demander avant de sauvegarder le dictionnaire, en utilisant le mot AUTORUN.
Ce mot s'utilise sous la forme :

AUTORUN « nom »

Donnons un exemple :
Fabriquons d'abord un mot, au hasard...
: DICO (---) BELL BELL RED FRAME VLIST ;
Puis :
AUTORUN DICO
Mettez votre LEP en enregistrement, puis :
23 SAVE

Maintenant, éteignez votre ordinateur car SAVE protège le dictionnaire.
Rallumez votre unité centrale, et mettez votre magnétophone en lecture.
Puis :
23 RUN ...
Le dictionnaire revient dans l'état où il était auparavant, et le mot DICO s'exécute.

Si vous possédez un MO5, cette option de chargement automatique ne sera possible qu'à partir d'une disquette (voir Manuel de référence).

Chaque dictionnaire ainsi sauvegardé étant numéroté, vous pouvez conserver sur une même cassette plusieurs applications, et charger celle que vous voulez. Il est prudent de noter soigneusement les numéros du compteur du magnétophone où vous avez commencé vos sauvegardes, et aussi de sauvegarder plusieurs fois vos applications ou vos écrans (En général, 2 fois suffisent...).
Lorsque vous voulez par la suite sauter une de vos sauvegardes, vous pourrez utiliser la commande suivante :

SKIP (---)
Positionne la tête de lecture du magnétophone à la fin du prochain fichier de la cassette.
Affiche le nom du fichier non chargé.

En fait, SKIP saute n'importe quel fichier, qu'il soit une sauvegarde dictionnaire, un écran de l'éditeur, ou même un programme ou un fichier BASIC.

B-2-2 La gestion par « blocks »

En ce qui concerne les écrans de l'éditeur, ils seront sauvegardés sur cassette dans un format fixe de « blocks » (selon la terminologie classique du FORTH) de 1024 octets.
On peut relire ces blocks depuis l'éditeur (par la commande CNT-R), mais aussi en dehors de l'éditeur, grâce à la commande :

BLOCK (n1 — n2)
Lit l'écran numéro n1, à moins qu'il soit déjà présent en mémoire, et laisse sur la pile l'adresse n2 du début de la zone où il est rangé dans votre unité centrale.

Le principal intérêt de ce mot, dans un premier temps, est de pouvoir utiliser les écrans non seulement pour contenir du texte à interpréter par FORTH, mais aussi pour conserver des données.

Exemple :
Répertoire téléphonique sur cassette (ou sur disquette).

Commençons par écrire notre répertoire en utilisant l'éditeur. Nous écrirons le nom et le prénom en 14 caractères chacun, et le numéro de téléphone en 12 caractères, de sorte que chaque ligne de l'écran contiendra les renseignements relatifs à un même individu.

```
10 EDIT
Dans l'écran 10 :
LEGRAND      Alexandre   11-111.33.44
QUATORZE     Louis        2- 33.56.78
LUPIN        Arsène      22- 22.22.22
VIVE         Forth       99-999.99.99
SEFI         Société     40- 35.60.86
Etc.
ZZZZ         (Nom inexistant)
Ensuite :
STOP         pour sortir de l'éditeur.
10 BLOCK C/L @   Laisse sur la pile l'adresse et la longueur de la première ligne
                 de l'écran.
TYPE  LEGRAND  Alexandre  11-111.33.44  OK
```

Utilisation :
Nous allons fabriquer un mot qui donne le numéro de téléphone lorsqu'on connaît le nom...
Le nom sera rangé dans une chaîne de caractères NOM.
Nous utiliserons deux nouveaux mots :

—TRAILING (str1 — str2)
qui modifie la longueur d'une chaîne de caractères en enlevant les blancs de la fin.

(LINE) (n1 n2 — str)
qui laisse sur la pile l'adresse et la longueur de la ligne n1 de l'écran n2.

Exemple :

```
" ABCD " TYPE ABCD  OK                                  ( longueur 9)
" ABCD " -TRAILING TYPE ABCD  OK                        ( longueur 4)
1 10 (LINE) 14 LEFT$ -TRAILING TYPE QUATORZE  OK
```

(La première ligne est numérotée 0, comme pour les tableaux)
Et maintenant, allons-y...
Nous supposons que vous avez tapé votre répertoire en utilisant l'éditeur, en commençant dans l'écran 10, et en continuant dans les suivants (11, 12, ...). Le dernier nom tapé est ZZZZ, et ne pourra jamais être demandé (Il sert d'indicateur de fin de répertoire).

```
14 STRING NOM
1 CONSTANT TROUVE        1 CONSTANT FINI
VARIABLE BLOCK-REDP      VARIABLE LIGNE
```

Nous utiliserons le mot AU-SUIVANT, qui modifie les variables BLOCK-REP et LIGNE, lorsqu'on passe d'un nom au suivant. Si on n'est pas à la ligne 24, il suffit d'augmenter LIGNE de 1, sinon on passe au block suivant, à la ligne 0.

```
: AU-SUIVANT                        ( ---)
    LIGNE @
    24 <                            ( Ligne 0 à 23 ???)
    IF 1 LIGNE +!                   ( Si oui, la ligne suivante)
    ELSE                            ( Sinon ...)
        1 BLOCK-REP +!              ( Block suivant)
        0 LIGNE !                   ( Ligne 0)
    THEN ;
```

Pour utiliser la ligne LIGNE du block BLOCK-REP, nous utiliserons :

```
: LIGNE$                            ( --- str)
    LIGNE @ BLOCK-REP @ (LINE) ;
```

Le mot suivant, RECHERCHE/NOM (Recherche par nom) laisse sur la pile la valeur TROUVÉ (1) ou NON-TROUVÉ (0) suivant que le nom contenu dans la variable NOM figure au répertoire ou non. Dans le cas où le nom est trouvé, les variables BLOCK-REP et LIGNE auront les valeurs convenables...

```
: RECHERCHE/NOM                     ( — n)
10 BLOCK-REP ! 0 LIGNE !            ( Début du répertoire)
BEGIN
    LIGNE$ 12 LEFT$-TRAILING        ( Nom lu dans le répertoire)
    DDUP NOM $=                     ( Est-ce le nom cherché ?)
    IF DDROP TROUVE                 ( Oui !!!)
        FINI                        ( Pour terminer la boucle)
    ELSE " ZZZZ" $=                 ( Est-ce le dernier ????)
        IF TROUVE NOT               ( Si oui, nom inconnu...)
            ." n'est pas au répertoire"
            FINI                    ( Recherche terminée)
        ELSE                        ( Sinon)
            AU-SUIVANT              ( Nom suivant du fichier ?)
            FINI NOT                ( La recherche doit continuer)
        THEN
    THEN
UNTIL ;                             (FINI, ou pas FINI ?)
```

Voilà, il ne reste plus qu'à faire travailler nos mots. Faisons par exemple un mot qui donne le numéro de téléphone d'une personne lorsqu'on connaît son nom :

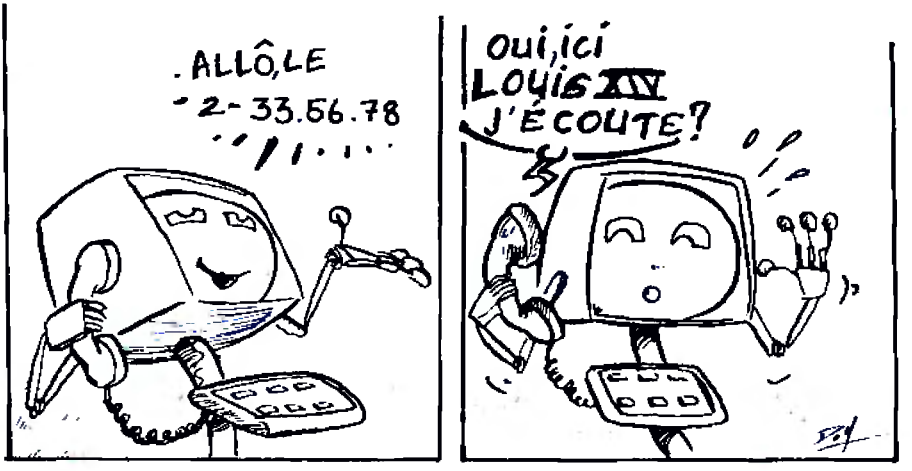
```
: TEL?     ( ---)
    ." Nom : " INPUT$          ( Demande un nom)
    NOM $!                     ( Le met dans la variable NOM)
    RECHERCHE/NOM              ( TROUVÉ, ou pas TROUVÉ ??)
    IF                         ( Si trouvé...)
        LIGNE$ 28 +            ( Début du numéro de téléphone)
        28 12 MID$ TYPE        ( Affiche le numéro)
    ELSE BELL                  ( Sinon, drelin ... )
    THEN ;
```

Quel est le numéro de téléphone de Louis QUATORZE ?

```
TEL?     Nom : QUATORZE     2-33.56.78  OK
```

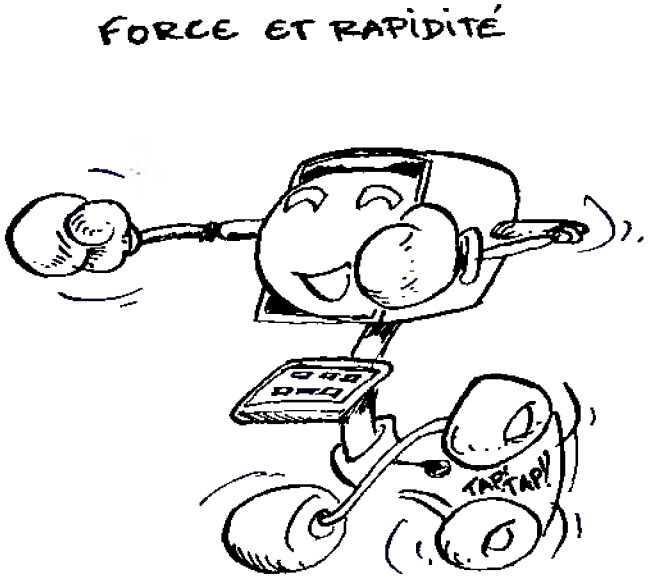
Ou :

```
TEL?     Nom : QUINZE       n'est pas au répertoire.
```



Annexe C

FORTH, c'est aussi bien d'autres choses



Ce manuel est un manuel d'initiation, et son contenu a été volontairement limité aux commandes indispensables pour une programmation élémentaire, mais confortable.
Son objectif est surtout de donner des idées et des méthodes d'écriture de vos applications, sans être trop compliqué, sans essayer d'être exhaustif...
Toutefois, FORTH est beaucoup plus souple et plus puissant que ce qui a été décrit ici. Il a des possibilités très vastes, que vous pouvez découvrir en étudiant le manuel de référence.
Dans cette annexe, nous allons essayer de donner des idées (très superficielles) de ces possibilités...

C-1 FORTH = souplesse + rapidité

FORTH est un langage d'une grande souplesse, en ce sens qu'il est à la fois un langage de haut niveau, disposant de mots de structuration très puissants, et un langage proche de la machine, permettant d'agir sur la mémoire avec beaucoup de facilité.

C-1-1 FORTH est un langage de haut niveau

Cela signifie qu'il permet une programmation « en langage clair ». Les spécialistes évaluent le niveau d'un langage à la puissance de ses instructions de structuration, (structuration des données et structuration de la programmation). Et dans ce domaine, FORTH est certainement un des mieux lotis...

Au point de vue structuration des programmes :
Nous avons étudié jusqu'ici les structures :
IF ... THEN
IF ... ELSE ... THEN
BEGIN ... UNTIL
DO ... LOOP

Les autres mots de cet acabit, qui font de FORTH un des langages les plus maniables de ce point de vue sont :
Sélection :
CASE (en liaison avec OF, ENDOF, ENDCASE)
GODO (en liaison avec THEN)

Répétitions :
DO ... +LOOP (analogue au FOR...STEP...NEXT de BASIC)
DO ... /LOOP (un +LOOP non signé)
Et surtout, le plus puissant :
BEGIN ... WHILE ... REPEAT

Tous ces mots figurent au Manuel de Référence, et vous ne devriez pas avoir de problème de mise en œuvre, ils sont suffisamment clairs en eux-mêmes.

Structuration des données
Là, votre FORTH de base vous semblera peut-être un peu plus pauvre, puisqu'il ne contient que des constantes, des variables, des chaînes de caractères, et des tables ou tableaux numériques ou alphanumériques. En outre, au point de vue numérique, il ne travaille qu'en nombres entiers...
Cette impression est trompeuse, il est très facile d'utiliser ces mots pour dé?? des types de données plus complexes.

Exemple :
Si vous voulez utiliser des données composées d'une chaîne de caractères longueur 10 et d'un nombre (en 2 octets), vous pouvez très bien vous servir d'une chaîne de chaîne de caractères (ou d'un tableau de chaînes) de longueur 12.

```
12 STRING DONNEE
```

Et des mots pour extraire de la chaîne DONNEE les informations souhaitées :

```
: NOM      ( str 1 --- str 2)    10 MIN ;
: NBRE     ( str --- adr)        DROP 10 + ;
```

Vous pourrez alors utiliser « DONNEE NOM » comme une chaîne de caractères, et « DONNEE NBRE » comme une variable ordinaire :

```
DONNEE NOM TYPE   et   " xxxxxx" DONNEE NOM $!
ou
nn DONNEE NBRE !   et   DONNEE NBRE @
```

Cet exemple est un peu artisanal (FORTH peut beaucoup mieux), mais il montre comment l'utilisation de mots adéquats vous permet de réaliser ce que d'autres langages font de manière plus ou moins cachée.

Par ailleurs, nous n'entrerons pas ici dans la polémique concernant les nombres entiers et décimaux (en fait entre virgule fixe et virgule flottante), utilisations scientifiques et industrielles de FORTH (astronomie, contrôle de processus, robotique, etc.) suffisent à défendre son point de vue.

L'idée générale est que les calculs en nombres entiers sont plus rapides, et que même si la programmation est un plus ardue, en général (!!!) on ne programme qu'une fois alors qu'on exécute le programme plusieurs fois. L'efficacité est à rechercher lors de l'exécution plutôt que de lors de l'écriture de l'application...

C-1-2 FORTH est un langage de bas niveau

Il est très facile en FORTH de consulter la mémoire ou de la modifier. Les mots de base sont :

C! (octet adr ---)
range l'octet à l'adresse adr

et

C@ (adr --- n)
n est le nombre 16 bits dont l'octet de poids faible est l'octet à l'adresse adr. L'octet de poids fort de n vaut Ø.

Voulez-vous jeter un coup d'œil à votre dictionnaire, ou à une autre partie de la mémoire de votre ordinateur ??? Rien de plus facile, utilisez le mot DUMP.

DUMP (adr n —)
Affiche les contenus des n octets (ou un peu plus, DUMP affiche 8 octets à la fois) qui commencent à l'adresse adr. A gauche, affichage des valeurs en hexadécimal, et à droite affichage des caractères ASCII correspondants.

Exemple :
Définissons la chaîne de caractères MACHIN:

```
1Ø STRING MACHIN        " ABCDEFGHI" MACHIN $!
```

Puis, pour voir ce qui est en mémoire, tapez fidèlement les commandes suivantes (qui seront expliquées dans la suite) :

```
' MACHIN 1Ø DUMP
```

L'affichage donne :

```
63EA: A 9 41 42 43 44 45 46 ..ABCDEF
63F2: 47 48 49 Ø 4 44 45 4D GHI..DUM
```

Vous pouvez lire facilement dans la partie droite (affichage ASCII) le contenu de la chaîne (adresses hexadécimales 63EC à 63F5). Le code A (1Ø en hexa) à l'adresse 63EA représente la longueur maximum de la chaîne, et le 9 qui suit représente sa longueur actuelle.

Vérification :
MACHIN HEX . . 9 63EC OK
Petite expérience :
HEX 41 63F2 C! (Remplace G par A)
MACHIN TYPE ABCDEFAHI OK

Le mot **CALL** vous permet d'appeler un sous-programme écrit en langage machine (terminé par l'instruction RTS), que vous pourriez par exemple ranger dans un block. Voir dans le Manuel de Référence les mots CALL, REGD, REGX, REGY, REGU, et REGS.

Il est par ailleurs possible d'écrire des mots FORTH directement en langage machine. Nous n'en dirons pas plus dans ce Manuel d'Initiation...

C-1-3 La rapidité et FORTH

Si vous avez fait des comparaisons, vous avez certainement constaté que FORTH est très rapide. On reconnaît que FORTH est en moyenne 3 à 5 fois plus lent qu'un langage machine, mais 3 à 10 fois plus rapide que le BASIC (dans une application complexe, pas pour une seule instruction...).
Les raisons essentielles sont simples à concevoir (outre le fait qu'il travaille sur des nombres entiers) :

— FORTH utilise une pile pour ranger les paramètres des mots qu'il utilise. Cela a pour principale conséquence que les mots FORTH n'ont pas à chercher où sont cachés leurs paramètres, ils savent tout de suite où aller les chercher. C'est aussi pour cette raison que FORTH utilise la notation polonaise postfixée...

— FORTH ne fait que très peu de vérifications. A vous de ne pas faire d'erreurs de programmation. Il faut savoir ce qu'on fait passer en priorité... Ou bien vous voulez un langage qui vérifie tout à votre place (et il faudra lui laisser le temps de faire les vérifications), ou bien vous voulez quelque chose de rapide, auquel cas il convient de le décharger des besognes inutiles (si vos programmes sont bien écrits).
Par exemple, si vous tapez en FORTH : 1 0 / (diviser 1 par 0) vous n'aurez aucune protestation, aucun message d'erreur. Mieux que cela (ou pire suivant votre goût), le résultat de l'opération sera – 1 !!!
Simplement parce que – 1 s'écrit 65535 en non signé, et que c'est le plus grand nombre représentable en 2 octets...
La plupart des autres langages, ici, auraient commencé par vérifier si le diviseur est 0, et vous auraient envoyé un message du genre « Erreur. Division par zéro ». Mais cette vérification se paie en temps d'exécution...

C-2 Une idée du fonctionnement de FORTH

Il n'est pas question ici de décrire complètement le fonctionnement du langage. Nous allons seulement voir comment sont constitués les mots du dictionnaire, et donner des idées sommaires sur les différents types de mots du FORTH...

C-2-1 Structure d'un mot du dictionnaire

Chaque mot est rangé au dictionnaire en deux parties consécutives :
Une partie « En-tête », et une partie « Paramètres ».
Le mot ' (apostrophe) donne l'adresse du début de la zone des paramètres d'un mot. Cette adresse étant connue, le mot NFA donne l'adresse du début de la zone En-tête.

Reprenons notre exemple du C-1-2. Nous avons éteint et rallumé notre TO-7, puis nous avons tapé :

```
10 STRING MACHIN " ABCDEFGHI" MACHIN $! OK
' MACHIN NFA 20 DUMP
```

On obtient :

```
63DF: 86 4D 41 43 48 49 CE 3F éMACHI#?
63E7: EB 1E 56 A 9 41 42 43     #.V..ABC
63EF: 44 45 46 47 48 49 0 4     DEFGHI..
```

(A l'écran, les caractères représentés par un # sont des caractères graphiques).
La zone En-tête débute à l'adresse 63DF, et se termine à l'adresse 63E9. Elle débute par le nom du mot (86 signifie entre autres « longueur 6 » ; 41, 42, etc. sont les codes ASCII des caractères du nom. Le dernier caractère a son bit 7 à 1, d'où un affichage ASCII perturbé).
La zone des paramètres commence à l'adresse 63EA. Elle se compose ici des codes des caractères qui constituent la chaîne de caractères précédés de 2 octets donnant la longueur maximum et la longueur actuelle.
La définition se termine à l'adresse 63F5 (fin actuelle du dictionnaire). Si par exemple on définit un autre mot, sa définition commencera à l'adresse suivante (en 63F6).
Vérification : XXXXX ; ' MACHIN NFA 30 DUMP (pour voir l'En-tête de XXXXX).

C-2-2 Interprétation Exécution ou Compilation

Nous avons dit au début de ce manuel que FORTH a deux modes de travail, le mode « Exécution » et le mode « Compilation » (Mise au dictionnaire). C'est bien vrai.

En fait, lorsque FORTH travaille, il interprète le flot d'entrée (clavier, ou écran de l'éditeur) pour en isoler les mots (séparés par des blancs), et à chaque fois qu'il a trouvé un mot XX, il teste la variable STATE pour savoir dans quel mode il est.

S'il est en mode « Exécution », il exécute le mot XX (nous supposerons que ce mot est un mot qu'il connaît).
S'il est en mode « Compilation », c'est qu'on est en train de définir un mot YY (entre : et ;). Dans ce cas, il se contente de mettre l'adresse du CF de XX dans le PF de YY, à la suite des autres paramètres.

Et si l'interpréteur rencontre par exemple le mot ":" ??? Que se passe-t-il ? Pas compliqué : le mot ":" est un mot qui signifie « Ce qui suit est une définition ».
Il faut donc :
— Fabriquer une en-tête pour le nom suivant.
— Passer en mode « Compilation » en positionnant STATE.
C'est le mot ";" qui repassera en mode « Exécution ».

Les autres mots de définition ont un fonctionnement analogue, à cela près qu'ils ne positionnent en général pas la variable STATE. En fait, les mots de définition ne sont pas autre chose que des compilateurs spécialisés (au sens du FORTH).
Lorsqu'on est en train de définir un mot YY en utilisant un mot de définition ZZ (qui peut être VARIABLE, CONSTANT, :, ou tout autre mot pouvant servir à définir des mots), c'est ZZ qui dit ce qu'il faut faire pour mettre XX (le mot actuellement traité par l'interpréteur) dans la définition de YY. C'est donc ZZ (sans préciser davantage pour le moment) qui est exécuté.

C-2-3 Fabriquer des « compilateurs »

Un des gros intérêts du FORTH, c'est qu'on peut aussi définir soi-même des mots de définition. En fait, FORTH est complètement extensible, et sa plus grande qualité n'est pas d'avoir un dictionnaire extensible, mais de pouvoir définir des structures (de programmation ou de données).qui ne sont pas prévues dans la version de base. Une étude complète de cette question dépasse le cadre d'un manuel d'initiation, nous pouvons en donner une idée succincte.
Un mot de définition ZZ doit avoir en réalité deux types d'action :
— Une action au moment de la fabrication d'un mot YY. C'est cette action qui dira par exemple (voir ci-dessus) comment procéder pour mettre un mot XX dans la définition de YY.
— ZZ doit aussi décrire comment s'exécutera le mot YY lorsque l'interpréteur FORTH le rencontrera.

Ces deux actions sont décrites en utilisant les mots **CREATE** et **DOES>**. Sans entrer dans les détails, voyons par exemple la définition du mot CONSTANT.

```
: CONSTANT      ( Définition du mot de définition)
      CREATE    ( Actions à exécuter lors de la définition de YY)
      ,         ( Mettre la valeur au dictionnaire)
      DOES>     ( Actions à exécuter lors de l'exécution de YY)
      @         ( Mettre la valeur sur la pile)
      ;         ( Et c'est fini, pas plus compliqué que cela...)
```

Précisons un peu :
1234 CONSTANT YY

C'est la partie « CREATE » qui s'exécute :
— Fabrication de l'en-tête du mot YY.
— Le nombre 1234 est mis au dictionnaire, dans le PF de YY.
YY . 1234 OK

C'est la partie « DOES> » de CONSTANT qui est exécutée par YY.
— Exécution du mot « @ » (adr --- n)
(Dépose simplement la valeur 1234 sur la pile)
C'est cette valeur qui est ensuite affichée par le mot « point ».
N'êtes-vous pas émerveillé par la simplicité de ce mécanisme ?
Bien entendu, toutes les définitions ne sont pas aussi simples, mais dès que vous savez clairement ce que doivent réaliser les deux étapes d'un mot de définition (à vous d'analyser votre problème), vous pouvez vous définir des structures non prévues. La plupart des mots de définition de votre FORTH sont écrits de cette façon, et rien ne vous empêche d'en définir d'autres (par exemple « point-écran », « liste », etc.)
Cette méthode est aussi beaucoup plus satisfaisante pour l'esprit que la méthode présentée au C-1-1 pour définir des structures « complexes ».

C-2-4 On exécute, ou on compile ?

Il est des mots qui agissent sur la compilation. C'est en particulier le cas de tous les mots qui servent à structurer vos programmes (IF, ELSE, THEN, DO, LOOP, BEGIN, UNTIL, WHILE, REPEAT, CASE, OF, ENDOF, ENDCASE, GODO, +LOOP, /LOOP, et j'en oublie peut-être).
Ils doivent donc **s'exécuter pendant la compilation** d'autres mots définis par :...;
On les appelle pour cette raison « mots immédiats » (Leur exécution n'est pas mise en attente, FORTH les exécute dès qu'il les rencontre).

Si vous voulez définir vous aussi des mots de structuration (Si, si, vous pouvez le faire !), il faudra que vous les définissiez comme étant immédiats. Cela se fait tout simplement en les définissant comme d'habitude, puis, avant de passer à une autre définition, en tapant le mot **IMMEDIATE**.

Donnons un exemple sans intérêt, mais démonstratif :
: XXX ." Coucou, je suis là... " ; IMMEDIATE OK
XXX Coucou, je suis là... OK
Si maintenant nous utilisons XXX dans une définition :
: YYY ." Moi aussi." XXX ; Coucou, je suis là... OK
Et à l'utilisation de YYY :
YYY Moi aussi. OK

Compris ? XXX a été exécuté lors de la compilation de YYY, et n'a pas été compilé. Il ne fait pas partie de la définition de YYY.

Si un mot est immédiat, on peut malgré tout obliger FORTH à le compiler dans une définition, en le prévenant juste avant ce mot (sinon il sera trop tôt ou trop tard) par le mot **[COMPILE]**.

Exemple :

```
: YYY2 [COMPILE] XXX ." Moi aussi" ;  OK
YYY Coucou, je suis là... Moi aussi.  OK
```

Autre exemple :
Le mot ' est un mot immédiat en ce sens qu'il dépose sur la pile le PFA du mot qui le suit dans le flot d'entrée.

```
' XXX NFA 30 DUMP fait un dump du mot XXX (voir plus haut)
```

Si on utilise dans une définition :

```
: DUMP-MOT     (---)     ' NFA 30 DUMP ;
```

ne pourra pas s'utiliser sous la forme DUMP-MOT XXX, car le flot d'entrée **au moment de la compilation** est la définition elle-même. Autrement dit, le mot DUMP-MOT comme il est écrit ne dumpera jamais que le mot NFA (mot suivant).
Pour que DUMP-MOT puisse faire le dump de n'importe quel mot, il faut le définir comme suit :

```
: DUMP-MOTS     (---)     [COMPILE] ' NFA 30 DUMP ;
```

Vous pourrez alors taper « DUMP-MOTS XXX » pour faire un dump de 30 octets à partir du début de la définition du mot XXX.

Mieux ! On peut rendre des mots immédiats, même s'ils ne sont pas prévus comme étant immédiats (Où donc cela va-t-il s'arrêter ???).
Il suffit, en cours de définition, de mettre le mot [pour que les mots suivants soient immédiatement exécutés, et non compilés. Et cela jusqu'à ce que l'interpréteur FORTH rencontre le mot].
Cela peut s'avérer très utile dans certaines situations.

Exemple :
Supposons que dans une définition on ait besoin d'utiliser des nombres donnés l'un en décimal et l'autre en hexadécimal... Pour les besoins de l'exemple, ajoutons 765 (décimal) à A6B7 (hexa).
Une solution (naturelle, mais fatigante) consisterait à calculer les 2 nombres en décimal ou en hexa, puis de les ajouter.
Une mauvaise solution serait de définir (supposons que nous soyons en décimal au moment de la définition) :

```
: WWW 765 HEX A6B7 + . ; A6B7 ? UNKNOWN  OK
```

Que s'est-il passé ?
Le mot HEX (qui n'est pas immédiat) a été compilé dans la définition de WWW, mais n'a pas été exécuté. Autrement dit, lorsque FORTH rencontre le mot A6B7, il est toujours en décimal et ne connaît pas ce mot, sauf si par hasard vous l'aviez défini auparavant.
Si au lieu de A6B7 nous avions pris un nombre interprétable en décimal, par exemple 1234, le résultat aurait été 1999 (somme en décimal), mais lors de l'exécution de WWW la base 16 aurait été positionnée. De sorte que l'affichage final aurait été : 7CF OK (Ecriture hexadécimale du nombre 1999).
Comment faire ?

C'est tout bête, il suffit de passer en base 16 pendant la compilation. Cela donne :

```
DECIMAL
: WWW1    765       (on est en base 10)
          [HEX]     (sortie provisoire du mode compilation)
          A6B7      (on est en base 16)
          + U. ;
```

Nous pouvons d'ailleurs vérifier que nous sommes en hexa, exactement comme si nous avions tapé HEX en dehors de la définition... (HEX a été immédiatement exécuté).

```
10 1 - . F  OK    (une manière comme une autre de vérifier la base)
DECIMAL           (pour revenir en base 10)
WWW1 43444  OK
```

Autre exemple :

Pour mettre le bit 5 à 1 en cours de définition.

```
: ....            (début de la définition)
    [ 2 BASE ! ]
    00100000      (compris en binaire)
    [ DECIMAL ]
...               (Les nombres suivants seront lus en décimal)
```

N'est-ce pas extraordinaire ?

Il y aurait encore beaucoup à dire sur le fonctionnement du FORTH et sur les autres bonnes surprises qu'il contient.

Dans cette annexe, nous n'avons fait que survoler quelques unes des particularités qui font de FORTH un des langages les plus ouverts, les plus riches, les plus agréables à étudier et à manipuler.

Vous avez maintenant les moyens de parcourir et de comprendre la plupart des mots du Manuel de Référence. C'est bien là l'objectif de ce manuel d'initiation.

Annexe D

Codes ASCII

Décimal	Hexa	Touche(s) à appuyer	Action éventuelle sur l'affichage (voir EMIT)
Caractères de contrôle			
0	0		
1	1	CNT-A donne le code 91	
2	2	STOP ou CNT-B	
3	3	CNT-C	
4	4	CNT-D	
5	5	CNT-E	
6	6	CNT-F	
7	7	CNT-G	Sonnette
8	8	← ou CNT H	Recul du curseur (d'un caractère)
9	9	→ ou CNT-I	Avance du curseur
10	A	↓ ou CNT-J	Saut de ligne vers le bas
11	B	↑ ou CNT-K	Saut de ligne vers le haut
12	C	RAZ ou CNT-L	Effacement de l'écran
13	D	ENTREE ou CNT-M	Passage à la ligne
14	E	CNT-N	Mode semi-graphique (TELETEL) *
15	F	CNT-O	Mode alphanumérique (Fin TELETEL) *
16	10	CNT-P	
17	11	CNT-Q	Curseur visible
18	12	CNT-R	Répétition du dernier caractère * Envoi de 65, 18, 4 : "AAAAA OK"
19	13	CNT-S	
20	14	CNT-T	Curseur invisible
21	15	CNT-U	
22	16	ACC ou CNT-V	Voir les caractères spéciaux
23	17	CNT-W	
24	18	CNT-X	Efface la fin de ligne
25	19	CNT-Y	
26	1A	CNT - Z donne le code 93	
27	1B		Début de séquence d'attribution de couleur...
28	1C	INS	
29	1D	EFF	Effacement d'un caractère
30	1E	⇦	Curseur en début d'écran
31	1F		Début de séquence de positionnement du curseur ou de format d'écran.

* Sur TO7 seulement

Caractères affichables

32	20	Barre d'espacement	Espace
33	21	Maj-!	!
34	22	Maj-"	"
35	23	Maj-#	#
36	24	Maj.-$	$
37	25	Maj-%	%
38	26	Maj-&	&
39	27	Maj-'	'
40	28	Maj-(	(
41	29	Maj-)	)
42	2A	*	*
43	2B	+	+
44	2C	,	,
45	2D	-	-
46	2E	.	.
47	2F	/	/
48	30	0	0
49	31	1	1
50	32	2	2
51	33	3	3
52	34	4	4
53	35	5	5
54	36	6	6
55	37	7	7
56	38	8	8
57	39	9	9
58	3A	Maj-:	:
59	3B	Maj-;	;
60	3C	Maj-<	<
61	3D	Maj-=	=
62	3E	Maj->	>
63	3F	Maj-?	?
64	40	@	@

En mode majuscules uniquement

65	41	A	A
66	42	B	B
67	43	C	C
68	44	D	D
69	45	E	E
70	46	F	F
71	47	G	G
72	48	H	H
73	49	I	I
74	4A	J	J
75	4B	K	K
76	4C	L	L
77	4D	M	M
78	4E	N	N

79	4F	O	O
80	50	P	P
81	51	Q	Q
82	52	R	R
83	53	S	S
84	54	T	T
85	55	U	U
86	56	V	V
87	57	W	W
88	58	X	X
89	59	Y	Y
90	5A	Z	Z
91	5B	CNT - A	[
92	5C		
93	5D	CNT - Z	]
94	5E	Maj-@	Flèche vers le haut
95	5F		□
96	60		□

En mode minuscules uniquement

97	61	a	a
98	62	b	b
99	63	c	c
100	64	d	d
101	65	e	e
102	66	f	f
103	67	g	g
104	68	h	h
105	69	i	i
106	6A	j	j
107	6B	k	k
108	6C	l	l
109	6D	m	m
110	6E	n	n
111	6F	o	o
112	70	p	p
113	71	q	q
114	72	r	r
115	73	s	s
116	74	t	t
117	75	u	u
118	76	v	v
119	77	w	w
120	78	x	x
121	79	y	y
122	7A	z	z
123	7B		{
124	7C		□

125	7D		}
126	7E		□
127	7F		□

Caractères spéciaux
En mode minuscules uniquement

128	80	ACC + c + c	ç
129	81	ACC + Ø + a	à
130	82	ACC + Maj-' + a	á
131	83	ACC + Maj-@ + a	â
132	84	ACC + Maj-" + a	ä
133	85	ACC + Ø + e	è
134	86	ACC + Maj-' + e	é
135	87	ACC + Maj-@ + e	ê
136	88	ACC + Maj-" + e	ë
137	89	ACC + Ø + i	ì
138	8A	ACC + Maj-' + i	í
139	8B	ACC + Maj-@ + i	î
140	8C	ACC + Maj-" + i	ï
141	8D	ACC + Ø + o	ò
142	8E	ACC + Maj-' + o	ó
143	8F	ACC + Maj-@ + o	ô
144	90	ACC + Maj-" + o	ö
145	91	ACC + Ø + u	ù
146	92	ACC + Maj-' + u	ú
147	93	ACC + Maj-@ + u	û
148	94	ACC + Maj-" + u	ü
149	95	ACC + Ø + n	ñ
150	96	ACC + Maj-' + n	ń
151	97	ACC + Maj-@ + n	ñ
152	98	ACC + Maj-" + n	ñ
153	99	ACC + {	ß
154	9A		} caractères graphiques
...	...		définis par l'utilisateur
255	FF		}

Annexe E

Messages d'erreur

BA ERROR (Bad argument)
Réservé pour une utilisation future.

CD ERROR (Conditionals not paired)
Signale que lors de la dernière compilation, FORTH a rencontré un mot de fin de structure conditionnelle ou répétitive, mais n'a pas trouvé de mot de début de structure.
Exemples :
ELSE sans IF
THEN sans IF ou GODO
LOOP sans DO
UNTIL sans BEGIN ...
Signale aussi un mauvais appariement des mots de structuration. Par exemple une structure du genre DO...UNTIL, qui n'existe pas en FORTH.
Oubliez la dernière définition du dictionnaire, et refaites-la.

CO ERROR (Compilation only)
Le mot ne peut être employé qu'en mode compilation, dans une définition par :...; C'est le cas pour tous les mots de structuration, comme IF, THEN, DO, etc.

DF ERROR (Dictionnary Full)
Il n'y a plus de place pour créer un nouveau mot au dictionnaire. Peut-être avez-vous utilisé trop de buffers. Dans ce cas, oubliez le nom en cours de définition, sauvegardez vos écrans, diminuez le nombre de buffers, et recompilez votre application.
Le nombre de buffers minimum est 2.

DL ERROR (Delimiter not found)
Le flot d'entrée ne contient pas le délimiteur cherché.

DR ERROR (Disk range)
Concerne la gestion des disquettes en blocs physiques. Le numéro de bloc ne correspond pas à un lecteur de disquettes présent (Attention à la base utilisée pour les nombres).

ES ERROR (Empty Stack)
La pile des données est vide.
Si une erreur de ce type survient lors de l'exécution d'un mot, la vérification ne se fait qu'à postériori, à la fin du mot. Ce qui signifie que le mot s'exécute d'abord, utilisant les données qui sont sous la pile.

FS ERROR **(Full stack)**

La pile des données est saturée (contient plus de 64 nombres).
Si une erreur de ce type provient lors de l'exécution d'un mot, la vérification n'a lieu qu'à postériori. Votre mot continuera donc d'abord à s'exécuter, empilant éventuellement des données hors de la pile, dans la zone des variables système.
Les résultats peuvent être extrêmement curieux...

I> ERROR **(Input > 255 characters)**

Vous avez essayé de définir une chaîne de caractères de longueur supérieure à 255.
Ce message concerne les chaînes définies par '' ou ."", mais aussi les remarques entre parenthèses. Il vous manque probablement un guillemet ou une parenthèse fermante pour terminer une chaîne de caractères ou une remarque.

IO ERROR

Erreur d'entrée-sortie. Ce message concerne toutes les communications entre l'ordinateur et l'imprimante, les cassettes, ou les disquettes. Il peut indiquer aussi bien une erreur en lecture qu'en écriture.

LO ERROR **(Load only)**

Réservé pour une utilisation ultérieure.

NU ERROR **(Not unique)**

Le nom du dernier mot défini figure déjà au dictionnaire. Ce message n'est qu'un avertissement. Les mots déjà compilés en utilisant la version précédente ne sont pas affectés, mais seule la dernière version sera prise en considération pour les définitions ultérieures, tant que vous ne l'aurez pas « oubliée ».

PR ERROR **(Protected word)**

Le mot que vous essayez d'oublier est protégé. Il fait partie du dictionnaire de base (en cartouche ROM), ou a été sauvegardé sur cassette et rechargé par le mot RUN. Dans ce cas, vous pouvez le déprotéger en agissant sur la variable FENCE (Par exemple : 0 FENCE !).

TB ERROR **(Too big)**

Réservé pour une utilisation future.

UF ERROR **(Unfinished structure)**

Lors de la dernière compilation, une structure conditionnelle ou répétitive n'a pas été terminée. Par exemple DO sans LOOP, IF sans THEN, etc. Oubliez la dernière définition du dictionnaire, et refaites la.

UK ERROR **(UnKnown name)**

Le mot ne figure pas dans le dictionnaire (dans son état actuel, c'est-à-dire dans le vocabulaire actuellement utilisé). Peut-être l'avez vous mal orthographié... D'autre part, attention aux caractères de contrôle, invisibles à l'écran mais interprétés par FORTH. Exemples : En mode direct, utilisation de la flèche à droite au lieu de l'espace, de la flèche à gauche au lieu

de la touche EFF, de la touche RAZ pour effacer l'écran, etc. Ou encore, rebond sur la touche STOP à la sortie de l'éditeur...

VF ERROR (Vocabulary full)
Il n'est pas possible de créer un vocabulaire supplémentaire. Vous ne pouvez avoir en mémoire plus de 4 vocabulaires simultanément, FORTH et EDITOR inclus.

XO ERROR (Execution only)
Signale que le mot ne peut être employé qu'en dehors d'une définition par :...;

Annexe F

Solutions des exercices

1-a
```
: PAVE    INVCOLOR SPACE INVCOLOR ;
: BARRE   INVCOLOR 5 SPACES INVCOLOR ;
: LIGNE1  BARRE SPACE BARRE SPACE BARRE        ( 1re ligne de F O R)
          SPACE BARRE SPACE PAVE 3 SPACES PAVE ;        ( T et H)
: LIGNE2  PAVE 5 SPACES PAVE 3 SPACES PAVE SPACE PAVE 3
          SPACES PAVE
          3 SPACES PAVE 3 SPACES PAVE 3 SPACES PAVE ;
: LIGNE3  BARRE SPACE PAVE 3 SPACES PAVE SPACE BARRE
          3 SPACES PAVE 3 SPACES BARRE ;
: LIGNE4  PAVE 5 SPACES PAVE 3 SPACES PAVE SPACE PAVE 2
          SPACES PAVE
          4 SPACES PAVE 3 SPACES PAVE 3 SPACES PAVE ;
: LIGNE5  PAVE 5 SPACES BARRE SPACE PAVE 2 SPACES PAVE PAVE
          3 SPACES PAVE 3 SPACES PAVE 3 SPACES PAVE ;
: '' FORTH'' CR LIGNE1 CR LIGNE2 CR LIGNE3 CR LIGNE4 CR LIGNE5 ;
```

Si vous voulez écrire d'autres mots en gros caractères, il est commode de remarquer qu'on écrit à chaque fois un certain nombre de pavés suivi d'un certain nombre d'espaces. Vous pouvez donc vous constituer une provision de mots utiles... Par exemple, en notant n&p le mot qui affiche n pavés et p espaces :

```
: PAVES INVCOLOR SPACES INVCOLOR ;          : 1&0  1 PAVES ;
: 1&1 1 PAVES SPACE ;      : 1&2 1&1 SPACE ;      : 1&3 1&2 SPACE ;
: 5&1 5 PAVES SPACE ;      : 1&4 1&3 SPACE ;      : 1&5 1&4 SPACE ;
: 5&3 5&1 2 SPACES ;       : 2&3 2 PAVES 3 SPACES ;    etc.
: LIGNE1   5&1   5&1          5&1          5&1   1&3   1&0 ;
: LIGNE2   1&5   1&3   1&1    1&3   1&3    1&3   1&3   1&0 ;
: LIGNE3   4&2   1&3   1&1    5&3          1&3   5&0       ;
: LIGNE4   1&5   1&3   1&1    1&2   1&4    1&3   1&3   1&0 ;
: LIGNE5   1&5   5&1          1&2   2&3    1&3   1&3   1&0 ;
```

La suite est la même que précédemment...

2-b
```
: CARRE   DUP * ;          : CARRE ? CARRE . ;
: CUBE    DUP CARRE * ;          : CUBE? CUBE . ;
31 CUBE?    29791  OK      32 CUBE?     -32768  OK
```

2-c
```
DDUP      a b --- a b a b        ( Essayez : 1 2 DDUP .S)
DUP DUP   a b --- a b b b        ( puis 1 2 DUP DUP .S)
```

2-d
```
a b ------ a b a ------ a b a b
   OVER      OVER                    X est synonyme de DDUP...
```

2-e a b c ----- b c a ----- c a b
ROT ROT
Y réalise une permutation circulaire en sens contraire de ROT.
C'est l'opération inverse de ROT (ROT ROT ROT laisse la pile inchangée)

2-f : ab-cd * ROT ROT * SWAP - ;
Ou encore (Voir >R et R> dans le Manuel de Référence...)
: ab-cd * >R * R> - ; (plus rapide que la première version)

2-g : ad-bc ROT ROT ab-cd ;

2-h : (a−b)2 - DUP * ; ou : (a−b)2 - CARRE ;

2-i : a2−b2 SWAP CARRE SWAP CARRE - ;
ou : a2−b2 SWAP DUP * SWAP DUP * - ;

2-j : 2×2+3×−5 DUP 2 * 3 + * 5 - ;
Ce mot est écrit en utilisant la méthode de Horner (pour calculer la valeur d'un polynôme). Par exemple : 2×2 + 3× −5 = (2×+3)× −5
On aurait de même : ×3 + 2×2 + 3× + 4 = ((×+2)× +3)× +4
Et le calcul s'écrirait en FORTH : DUP DUP 2 + * 3 + * 4 +

2-k : HMS 60 /MOD 60 /MOD
. ." h. " . ." mn. " . ." s." ;

2-l PI* (n1 --- n2) donne un résultat correct pour n1 inférieur à 1490.
1489 PI* donne n2 = 4679. La multiplication par 2 est donc licite.
CIRC donne donc un résultat correct dans les mêmes conditions que PI*
Cela serait faux si on avait défini CIRC par : CIRC1 2* PI* ;
Dans ce cas, c'est 2r qui aurait dû être inférieur à 1490.
CIRC1 ne donnerait un résultat correct que pour r inférieur à 745.
De la même façon, AIRE donne un résultat valable lorsque le carré de r est inférieur à 1490, c'est-à-dire lorsque r et inférieur à 39.

2-m Pour CIRC, il faut que 2×r×22/7 < 32768 d'où r < 5214
Pour AIRE, il faut que r×r×22/7 < 32768
Le carré de r doit être inférieur à 10426, ce qui donne r<103.
Noter que l'ordre des opérations n'est pas indifférent, car la division n'est pas exacte (division entière) alors que la multiplication l'est.
Les résultats seront meilleurs en terminant par la division...
: PI* 22 7 */ ; : CIRC 2* PI* ; : AIRE DUP * PI* ;

3-a : EX3A (n --- n est décimal, affichage en base 2 et 16)
CR ." ---------- En base 2 : " 2 BASE ! DUP .
CR ." ---------- En base 16 : " HEX . DECIMAL CR ;
Puis 2334 EX3A donne : Base 2 : 100100011110 Base 16 : 91E

Autres résultats :	..... Base 2	..... Base 16 .....
4567	1000111010111	11D7
34	100010	22
−546	−1000100010	−222
7680	1111000000000	1E00
32767	111111111111111	7FFF

3-b HEX 7FFF DECIMAL . 32767 OK

3-c ETAT-RESERVE (Version 1. voir aussi chapitre 9)

```
CLS CR CR CR ." BOITES EN RESERVE"
CR ." ----------"
CR ." Choucroute : " Ø RESERVE CONSERVES ?
CR ." Cassoulet  : " 1 RESERVE-CONSERVES ?
CR ." Couscous   : " 2 RESERVE-CONSERVES ?
CR ." Potée      : " 3 RESERVE-CONSERVES ?
CR ." Raviolis   : " 4 RESERVE-CONSERVES ?
CR CR ;
```

3-d

```
5 ARRAY RAYON-CONSERVES
: VENTE ( n p ---        n boîtes vendues de la rubrique p)
      SWAP NEGATE SWAP RAYON-CONSERVES +! ;
: CHOUCROUTE -           ( n ---) Ø VENTE ;
: CASSOULET -            ( n ---) 1 VENTE ;     etc.
```

N'oubliez pas de remplir le tableau RAYON-CONSERVES avant utilisation.

3-e

```
Ø CONSTANT CHOUCROUTE     1 CONSTANT CASSOULET
2 CONSTANT COUSCOUS       3 CONSTANT POTEE
4 CONSTANT RAVIOLIS
: R  M     ( n p --- Déplace n boîtes de la réserve au magasin)
    DDUP SWAP NEGATE SWAP RESERVE-CONSERVE +! ( Pris dans la
réserve)
    RAYON-CONSERVES +! ;                          ( Et mis en rayon...)
: M  R     ( n p --- Déplace n boîtes du magasin à la réserve)
    DDUP VENTE LIVRAISON ; ( Voir exercice 3-d et manuel & 3-3)
```

3-f

```
VARIABLE TOTAL A PAYER
: TOTAL-ARTICLE ( n p --- Ajoute au total le prix de n boîtes de)
                                                  ( l'article numéro p)
    DDUP VENTE                              ( Voir VENTE à l'exercice 3-d)
    PRIX-CONSERVES * DUP .                  ( Prix de l'article ... affiché ...)
    TOTAL-A-PAYER +! ;                      ( Mise à jour au total à payer)
: CHOUCROUTE ( n ---     n boîtes vendues)
    CR ." Choucroute : " DUP . ." boîtes ... " Ø TOTAL-ARTICLE ;
: CASSOULET  ( n --- )
    CR ." Cassoulet : " DUP . ." boîtes ... " 1 TOTAL-ARTICLE ;
etc. ( mots COUSCOUS, POTEE, RAVIOLIS)
: VENDU    ( --- )    Initialise TOTAL-A-PAYER) Ø TOTAL-A-PAYER ! ;
: TOTAL    ( --- )
    CR ." ----------"
    CR ." Total à payer ..... " TOTAL-A-PAYER ? ;
```

Avec une imprimante, modifier CHOUCROUTE, etc. de la manière suivante :

```
: CHOUCROUTE PRINTER CHOUCROUTE CONSOLE ;
```

(ou définissez un nouveau mot P-CHOUCROUTE...)

et : TOTAL PRINTER TOTAL CONSOLE ;

... Sans vous soucier des messages "CHOUCROUTE NU" etc.

Ardoises :

```
100 ARRAY ARDOISE  Ø CONSTANT Dupont  1 CONSTANT Durand  etc.
: EN-COMPTE    ( n ---      Ajouter le total au compte numéro n)
        TOTAL-A-PAYER @ SWAP ARDOISE +! ;
: COMPTE?      ( n ---      Consultation de l'ardoise numéro n)
        ARDOISE ? ;
```

```
: REGLEMENT-COMPTE      ( n ---      Réinitialise l'ardoise numéro n)
      0 SWAP ARDOISE ! ;
```

4-a PREPARER-BOITES ... sans changement

```
: RESERVE-INSUFFISANTE      ( p x ---      x boîtes de l'article p)
      DUP ROT RAYON-CONSERVES @ >
      IF DROP ." Commande impossible" BELL BELL BELL
      ELSE PREPARER-BOITES
      THEN ;
: COMMANDE-BOITES      ( n p ---      Commande de n boîtes numéro p)
      SWAP OVER RESERVE-CONSERVES @ -      (pile : p x ---      )
      DUP 0> IF RESERVE-INSUFFISANTE
             ELSE ." Pas de problème, les boîtes sont en réserve" DDROP
             THEN ;
```

Puis par exemple : : COMMANDE-CHOUCROUTE 0 COMMANDE-BOITES ;
: COMMANDE-CASSOULET 1 COMMANDE-BOITES ;
etc.

4-b Modifications par rapport à l'exercice précédent :
(En utilisant les mots de l'exercice 3-f)
Dans COMMANDE-BOITES :
Remplacer SWAP OVER par DDUP (mémoriser n)
Remplacer DDROP par DROP TOTAL-ARTICLE
Dans RESERVE-INSUFFISANTE :
Remplacer PREPARER-BOITES par DUP PREPARER-BOITES TOTAL-ARTICLE
Ajouter : : COMMANDE VENDU ; (Initialisation de TOTAL-A-PAYER)
On peut alors utiliser :
COMMANDE 1 COMMANDE-CASSOULET 3 COMMANDE-CHOUCROUTE ... TOTAL

4-c

```
: ORDRE      ( n1 n2 n3 ---      Affichage dans l'ordre)
      DDUP < IF SWAP THEN ( Range n2 et n3, le plus petit au-dessus)
      ROT DDUP < IF SWAP THEN ( Le plus petit des 3 est sur la pile...)
      . DDUP < IF SWAP THEN . . ;
      ( On l'affiche, puis on range les deux autres et on les affiche)
```

Meilleur :

```
: RANGE      ( n1 n2 --- n3 n4      n3 = Max      n4 = Min)
      DDUP < IF SWAP THEN ;
: ORDRE RANGE ROT RANGE . RANGE . . ;
```

RANGE peut d'ailleurs être défini sans utiliser IF ni THEN...

```
: RANGE DDUP MAX SWAP ROT MIN ;
```

6-a

```
a - 160 199 PSET      ou      159 199 PSET      ( nombre pair de points)
b - 319 100 PSET      ou      319  99 PSET      ( nombre pair de points)
c - 160 100 PSET  ou  159 100 PSET  ou  ou 160 99 PSET  ou  159 99 PSET
```

6-b

```
: SABLIER-H-B                    ( ---                    De haut en bas)
      CLS AXE-H 2* 0             ( Pour I variant de 0 à 199 = 2*AXE-H)
      DO  AXE-H I -              ( écart par rapport à l'axe vertical)
          AXE-V OVER -  I        ( x1 = AXE-V - écart ; y1 = I)
          ROT AXE-V +   I        ( x2 = AXE-V + écart ; y2 = I)
          LINE
      LOOP ;
```

Autre méthode, inspirée de la première partie de SABLIER...

```
: SABLIER-H-B-2                    ( ---                    De haut en bas)
        CLS AXE-H AXE-H NEGATE     ( Pour I variant de -100 à 100)
        DO  AXE-V I -              ( x1 ... I=0 pour la ligne du milieu)
            AXE-H I +                    ( y1 ... varie bien e 0 à 200)
            AXE-V 1 + OVER         ( x2 ... y2=y1 donné par le OVER)
            LINE
        LOOP ;
```

Pour remplir le sablier de bas en haut, il suffit de remplacer y1 et y2 (qui valent I dans SABLIER-H-B) par 200-I... Pour cela :
En lignes 4 et 5 de SABLIER-H-B, remplacer I par : AXE-H 2* I -
A partir de SABLIER-H-B-2, c'est encore plus facile...
En ligne 4, remplacer + par - pour que y1 varie de 200=100-(-100) à 0

6-c Par exemple :

```
: A 160 10 ; : B 240 190 ; : C 40 60 ; D 280 60 ; : E 80 190 ;
```

Puis : PAGE CLS A PSET B LINETO C LINETO D LINETO E LINETO A LINETO

6-d

```
: ETOILE PSET 5 0 DO LINETO LOOP ;
```

Puis : A E D C B A ETOILE ou A C E B D A ETOILE (ou les deux)

6-e

```
: LIGNE-BRISEE ROT ROT PSET 0 DO LINETO LOOP ;
```

6-f

```
2 ARRAY MEMO (Pour le point à mémoriser)
: POLY CLS ROT ROT DDUP 1 MEMO ! 0 MEMO ! PSET
        1 DO LINETO LOOP
        0 MEMO @ 1 MEMO @ LINETO ;
```

On peut faire l'économie du tableau MEMO en utilisant soit le mot PICK, soit les mots >R et R> (voir Manuel de Référence). On obtient :

```
: POLY1 CLS DUP 2* 1+ PICK              ( Abscisse du premier point)
        OVER 2* 1+ PICK                 ( Ordonnée du premier point)
        PSET 0 DO LINETO LOOP ;
: POLY2 CLS ROT ROT DDUP >R >R PSET
        1 DO LINETO LOOP
        R> R> LINETO ;
```

L'idée de POLY1 est d'aller chercher au milieu de la pile les coordonnées du premier point à tracer (qui doit être aussi le dernier).
L'idée de POLY2 est de sauvegarder le premier point effectivement tracé non pas dans un tableau, mais sur la pile des retours...
Ex : A B C D 4 POLY (ou POLY1, ou POLY2)
POLY (ou POLY2) trace d'abord D, puis le relie dans l'ordre à C, B, A, D
POLY1 trace d'abord le point A, puis le relie dans l'ordre à D, C, B, A
Ou encore, en utilisant la pile :

```
: POLY3 CLS >R DDUP PSET
        R> 1 DO DSWAP LINETO LOOP
        LINETO ;
```

POLY3 garde le premier point sur la pile, et l'échange, au fur et à mesure des LINETO, avec les points suivants.

Autre méthode, inspirée de la première partie de SABLIER...

```
: SABLIER-H-B-2                    ( ---                    De haut en bas)
        CLS AXE-H AXE-H NEGATE     ( Pour I variant de -100 à 100)
        DO  AXE-V I -              ( x1 ... I=0 pour la ligne du milieu)
            AXE-H I +                    ( y1 ... varie bien e 0 à 200)
            AXE-V I + OVER         ( x2 ... y2=y1 donné par le OVER)
            LINE
        LOOP ;
```

Pour remplir le sablier de bas en haut, il suffit de remplacer y1 et y2 (qui valent I dans SABLIER-H-B) par 200-I... Pour cela :
En lignes 4 et 5 de SABLIER-H-B, remplacer I par : AXE-H 2* I -
A partir de SABLIER-H-B-2, c'est encore plus facile...
En ligne 4, remplacer + par - pour que y1 varie de 200=100-(-100) à 0

6-c Par exemple :

```
: A 160 10 ; : B 240 190 ; : C 40 60 ; D 280 60 ; : E 80 190 ;
```

Puis : PAGE CLS A PSET B LINETO C LINETO D LINETO E LINETO A LINETO

6-d

```
: ETOILE PSET 5 0 DO LINETO LOOP ;
```

Puis : A E D C B A ETOILE ou A C E B D A ETOILE (ou les deux)

6-e

```
: LIGNE-BRISEE ROT ROT PSET 0 DO LINETO LOOP ;
```

6-f

```
2 ARRAY MEMO (Pour le point à mémoriser)
: POLY CLS ROT ROT DDUP I MEMO ! 0 MEMO ! PSET
        I DO LINETO LOOP
        0 MEMO @ 1 MEMO @ LINETO ;
```

On peut faire l'économie du tableau MEMO en utilisant soit le mot PICK, soit les mots >R et R> (voir Manuel de Référence). On obtient :

```
: POLY1 CLS DUP 2* 1+ PICK              ( Abscisse du premier point)
        OVER 2* 1+ PICK                 ( Ordonnée du premier point)
        PSET 0 DO LINETO LOOP ;
: POLY2 CLS ROT ROT DDUP >R >R PSET
        1 DO LINETO LOOP
        R> R> LINETO ;
```

L'idée de POLY1 est d'aller chercher au milieu de la pile les coordonnées du premier point à tracer (qui doit être aussi le dernier).
L'idée de POLY2 est de sauvegarder le premier point effectivement tracé non pas dans un tableau, mais sur la pile des retours...
Ex : A B C D 4 POLY (ou POLY1, ou POLY2)
POLY (ou POLY2) trace d'abord D, puis le relie dans l'ordre à C, B, A, D
POLY1 trace d'abord le point A, puis le relie dans l'ordre à D, C, B, A
Ou encore, en utilisant la pile :

```
: POLY3 CLS >R DDUP PSET
        R> 1 DO DSWAP LINETO LOOP
        LINETO ;
```

POLY3 garde le premier point sur la pile, et l'échange, au fur et à mesure des LINETO, avec les points suivants.

6-g
```
: TX  GR     ( n1 --- n2) 2* 2* 2* ;     ( Coordonnée texte  graphique)
: CADRE      ( haut bas gauche droite ---)
        ROT SWAP TX  GR 10 + 319 MIN          ( Abscisse de droite)
        SWAP TX  GR 10 + 199 MIN PSET         ( Coin en bas à droite)
        TX  GR 3 - 0 MAX                      ( Abscisse de gauche)
        SWAP TX  GR 3 - 0 MAX BOXTO ;         ( Coin en haut à gauche)
```
Exemple :
```
CLS 10 20 WINDOW HOME                         ( Pour être tranquille)
17 3 LOCATE ." MENU" HOME
18 5 LOCATE ." GENERAL" HOME
```
La première ligne du titre est à la ligne 3, la dernière à la ligne 5.
Le caractère le plus à gauche est à la colonne 17.
Le caractère le plus à droite est à la colonne 24...
Donc : 3 5 17 24 CADRE HOME

7-a Nous donnerons ici deux solutions...

1 - Une solution de l'exercice :
```
182 CONSTANT YM          40 CONSTANT X1          279 CONSTANT XM
(Dessin de l'écran)
TABLE TAB-NOIRES ( Ordonnées des touches noires d'une octave)
        3, 9, 15, 24, 30,
: TOUCHE-NOIRE     ( y --- Dessine deux touches noires sur les côtés)
        16 OVER X1 1- OVER 3 + BOXF     ( y = ordonnée de début)
        YM 1+ SWAP 303 OVER 3 + BOXF ;  ( des touches à dessiner)
: UNE-OCTAVE                            ( y --- Dessine les touches d'une octave)
        5 0 DO I TAB-NOIRES OVER+       ( y = ordonnée de début)
              TOUCHE-NOIRE              ( de l'octave à dessiner)
           LOOP
        0 OVER 319 OVER LINE        ( Un trait pour séparer les octaves)
        16 SWAP 21 + 303 OVER LINE ;      ( Trait pour séparer mi-fa)
: TOUCHES                   ( Dessine toutes les touches)
        5 0 DO I 36 * 3 +           ( Adresses de début des octaves)
              UNE-OCTAVE LOOP ;
: DESSIN                    ( Dessine le fond de l'écran)
        BLACK FRAME GLOBAL PAPER CYAN COLOR WHITE PAPER
        GCOLOR
        LOCAL INK BLUE COLOR MAGENTA GCOLOR( 4 couleurs fixées)
        CLS X1 0 XM YM 1+ PAPER BOXF          ( zone de pointage)
        INK 8 1 DO I X1 * 0 OVER YM LINE LOOP     ( zones verticales)
        62 0 DO X1 I 3 * XM OVER LINE LOOP      ( zones horizontales)
        BLACK GCOLOR TOUCHES ;
(Lire les coordonnées du point visé par le crayon optique)
: BOF-Y         ( y --- cond      La valeur y est-elle invalide ???)
        DUP 0< SWAP YM > OR ;
: BOF-X         ( x --- cond      La valeur x est-elle invalide ???)
        DUP X1 < SWAP XM > OR ;
: BOF ?         ( x y --- x y cond          Le point visé est-il hors de la)
        OVER BOF-X OVER BOF-Y OR ;          ( zone de pointage ???)
: LIRE          ( --- x y              Coordonnées d'un point visé, dans la)
        BEGIN INPUTPEN BOF ?                    ( zone de pointage...)
           IF DDROP 0       ( Si invalide, on recommence à lire)
           ELSE 1 THEN      ( Si lecture valide, 1 pour fin-boucle)
        UNTIL ;
```

```
(Jouer les notes)
TABLE DUREE     3, 6, 12, 24, 48, 96,                ( Durées pour DUR)
: DUR!!                    ( x --- Calcule et fixe la durée des notes à jouer)
        XI / 1- DUREE DUR ;
: NOTE!!    ( y --- Calcule et joue la note, en fonction de y)
        3 / 60 - NEGATE 12 /MOD     ( Convertit x, y en numéros de zones)
        DUP 5 = IF DDROP 12 4 THEN     ( do octave 6 ---> do+ octave 5)
        1+ OCT 1 + NOTE ;              ( joue la note désirée)
: JEU ( ---)
        DESSIN
        BEGIN LIRE SWAP DUR!! NOTE!!
              ?TERMINAL     ( Pour stopper, appuyer une touche tout en)
        UNTIL ;             ( jouant une note valide)
```

Cette application fonctionne comme prévu, mais la zone de saisie d'une note est trop petite pour que le résultat soit intéressant.
Il est d'un meilleur effet de diminuer le nombre de notes pour agrandir ces zones.
Pour vous récompenser de nous avoir suivis jusqu'ici, voici une autre méthode pour jouer de la musique au crayon optique...
(Dessin du clavier. Utilisons des tables X et Y pour les positions)

```
: REMPLIR-TABLE-X     ( ---)     18 0 DO I 16 * 24 + , LOOP ;
TABLE X     REMPLIR-TABLE-X
TABLE Y     40 DUP, 60 + DUP , 60 + ,
TABLE MITON   ( Positions du clavier sans touches noires, correspondant)
0, 3, 7, 10, 14, 17, ( aux écarts d'un demi-ton, mi-fa et si-do)
: NOIRES-FOND     ( Dessine le fond des touches noires, sera en jaune...)
        0 X  0 Y 17 X 1-  1 Y  BOXF ;
: NOIRES-TOUR     ( Le trait de droite des cadres sera à redessiner)
        18 0 DO I X 8 - 0 X MAX  0 Y
                I X 8 + 17 X MIN 1 Y 1- BOX
            LOOP ;
: BLANCHES-FOND     ( Dessine les fonds de touches blanches, effaçant les)
        0 X  1 Y  17 X 1-  2 Y  BOXF                ( touches noires qui)
6 0 DO  I MITON X 8 - 0 X MAX  0 Y                  ( correspondent à des)
        I MITON X 7 + 17 X 1- MIN 1 Y BOXF              ( demi-tons...)
    LOOP ;
: BLANCHES-TOUR ( Dessine les contours des touches blanches...)
        0 X  0 Y  17 X  2 Y  BOX                        ( contour général)
        17 1 DO I X 1 Y I X 2 Y LINE LOOP          ( séparer les blanches)
        6 1 DO I MITON X DUP 0 Y OVER 1 Y LINE ( Cas des 1/2 tons)
                8 - 0 Y OVER I Y 1- LINE             ( Retrace les traits)
            LOOP ;   ( de NOIRES-TOUR effacés par BLANCHES-FOND)
: CLAVIER     ( ---)
        GLOBAL PAPER RED COLOR CSROFF YELLOW GCOLOR INK
        BLACK GCOLOR
        PAPER NOIRES-FOND     INK NOIRES-TOUR
        PAPER WHITE GCOLOR BLANCHES-FOND INK BLANCHES-
        TOUR ;
(Obtention d'une lecture valide du crayon optique)
: YVALIDE? ( y --- cond) DUP 0 Y > SWAP 2 Y < AND ;
: XVALIDE? ( x --- cond) DUP 0 X > SWAP 17 X < AND :
: VALIDE?   ( x y — x y cond) OVER XVALIDE? OVER YVALIDE? AND :
```

```
: LIRE        ( --- x y coordonnées d'un point du clavier)
        BEGIN INPUTPEN VALIDE?
                IF 1
                ELSE DDROP Ø THEN
        UNTIL ;
(Récapitulation. Jouer les notes)
TABLE BLANCHES 49, 51, 53, 54, 56, 58, 6Ø,
TABLE ALTEREES 49, 5Ø, 5Ø, 52, 52, 53, 54, 55, 55, 57, 57, 59, 59, 6Ø,
                (par demi-touche du clavier)
: BLANCHE ( x --- Calcule octave et note. Partie basse du clavier)
        Ø X - 16 / 7 /MOD 2+ OCT BLANCHES ;
: NOIRE     ( x --- Idem, pour la partie haute du clavier)
        Ø X -  8 / 14 /MOD 2+ OCT ALTEREES ;
: JEU        ( --- Le mot de la fin...)
        CLS RED FRAME
        CLAVIER
        BEGIN LIRE
                1 Y > IF BLANCHE ( Partie basse du clavier)
                      ELSE NOIRE THEN ( Partie basse du clavier)
                      NOTE ?TERMINAL
        UNTIL
        GLOBAL CYAN PAPER COLOR INK INITSCR ;
```

Cette application ne permet pas de modifier la durée des notes, mais elle est d'une utilisation plus agréable que la précédente.
Par ailleurs, il est possible d'ajouter dans JEU un mot DUREE? qui par exemple scrute le clavier (voir INKEY), et modifie la durée suivant la touche enfoncée...

8-a Il est commode pour l'utilisation ultérieure d'employer des variables X et Y pour mémoriser la position du curseur au début du déplacement...

```
VARIABLE X   VARIABLE VX   VARIABLE TEMPS
VARIABLE Y   VARIABLE VY   VARIABLE NB-PAS   VARIABLE CAR
(Pour le caractère, il sera souvent commode d'utiliser une constante)
: XY-LOC ( --) X @ Y @ LOCATE ; ( Sans commentaire)
: BZZ ( --- Déplace CAR de NB-PAS. Part de x,y. Direction VX, VY)
    XY-LOC CAR @ EMIT                    ( Affiche un premier caractère)
    NB-PAS @ Ø                            ( répéter nb-pas fois l'action)
    DO TEMPS @ MILLISECONDES                                ( attendre)
        YX-LOC BL EMIT                          ( effacer le caractère)
        VX @ X +! VY @ Y +!                ( modifier x et y)
        XY-LOC CAR @ EMIT LOOP ;      ( retracer le caractère CAR)
```

Exemple d'utilisation :

```
1 CHARTAB OVNI OVNI HEX Ø Ø 3C FF 5A 3C Ø Ø DECIMAL 154 DEFGR
```

Déplaçons l'OVNI au hasard dans l'écran...

```
: V?          ( --- n)     3 RND 1- ;        ( VX et VY vaudront -1, Ø ou 1)
: NPMX      ( --- n déplacement maximum autorisé compte tenu de X et VX)
    VX @ DUP IF Ø> IF 39 X @ -                                  ( Si VX>Ø)
                                  ELSE X @      THEN          ( Si VX<Ø)
                          ELSE DROP 39 THEN ;                  ( Si VX=Ø)
: NPMY      ( --- n  déplacement maximum autorisé compte tenu de Y et VY)
    VY @ DUP IF Ø> IF 24 Y @ -
                              ELSE Y @      THEN
                          ELSE DROP 24 THEN ;
```

```
: NP?        ( --- n                    choix au hasard du nombre de pas...)
    VX @ VY @ OR
    IF  NPMX NPMY MIN 1+  RND
    ELSE Ø THEN ;
: INIT ( ---) 20 X ! 10 Y ! OVNI 154 CAR ! Ø TEMPS !
    BLACK COLOR CSROFF PAGE ;
: VROOM ( ---) BEGIN V? VX ! V? VY ! NP? ?DUP
                    IF NB-PAS ! 7 EMIT BZZ THEN
                    PTRIG
                UNTIL INITSCR ;
```

Essayez maintenant : VLIST INIT VROOM
Votre OVNI dévore l'un après l'autre les caractères affichés à l'écran.
Pour un OVNI « vrombissant », mettez un « 7 EMIT » dans la boucle de BZZ...

9-a Deux des trois exercices ne sont possibles qu'avec des mots de longueur impaire.

```
25 STRING MOT      " FORTH" MOT $!
: CRTAB      ( ---) CR 8 SPACES :                (Pour faire une marge)
: JEU1       ( ---)
    CR MOT LEN DUP Ø
    DO CRTAB DUP I - MOT ROT RIGHT$ TYPE
        MOT I LEFT$ TYPE
    LOOP
    KEY DDROP CR ;
: JEU2       ( ---)
    CR MOT LEN 1+ DUP 1              ( Longueur + 1 sert dans la boucle)
    DO DUP 1 - 1 MIN 1-                          ( valeurs Ø, 1, 2, 1, Ø)
        OVER 2/ OVER -        ( Nbre d'espaces, ou début de sous-chaîne)
        CRTAB DUP SPACES                ( Réalisation des décalages)
        SWAP 2* 1+          ( La pile contient les paramètres de ss-chaîne)
        MOT DSWAP MID$ TYPE
    LOOP
    KEY DDROP CR ;
: JEU3       ( ---)
    CR MOT LEN 1+ DUP 1
    DO DUP I - 1 MIN 1- OVER 2/ OVER - CRTAB SPACES
        2* 1+ MOT ROT                ( Nombre de caractères à afficher)
        I 2* MOT LEN > IF RIGHT$ ELSE LEFT$ THEN ( à droite ou à gauche)
        TYPE
    LOOP
    KEY DDROP CR ;
```

Puis par exemple : " ANTICONSTITUTIONNELLEMENT" MOT $!
Suivi de : CLS JEU1 JEU2 JEU3... (Tapez une touche entre deux jeux ...)
Note : (Anticipation sur le 9-7).
Pour être sûr que la chaîne de caractères « MOT » ne contient pas un nombre pair de caractères, on peut demander les entrées au clavier...

```
: MOT?       ( ---)  Ø. BEGIN CR . " Donne-moi un mot : "
                        DDROP INPUT$ DDUP LEN 2 MOD
                    UNTIL
                    MOT $! ;
```

Noter le Ø. qui permet le démarrage de la boucle (voir l'action du DDROP)

9-b
```
: $> = ( str1 str2 --- cond) $< NOT ;
: DDDUP ( d1 d2 --- d1 d2 d1 d2 ou encore)
        ( str1 str2 --- str1 str2 str1 str2) DOVER DOVER ;
: $>    ( str1 str2 --- cond)
        DDDUP $< S   D     (Transforme cond en nombre double longueur)
        DROT DROT $=                    (pour la manipulation de la pile)
        SWAP DROP OR NOT ;
```

Les manipulations de pile sont plus simples en se servant de >R et R>. (voir Manuel de Référence) On obtient alors :

```
: $>    ( str1 str2 --- cond)    DDDUP $< >R $= R> OR NOT ;
```

9-c Le plus simple est de prendre une variable chaîne auxiliaire.

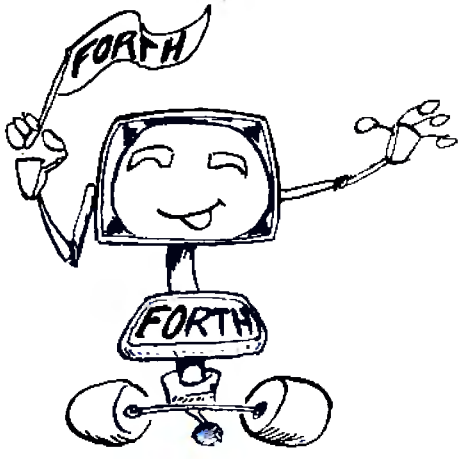
```
255 STRING MOT-A-REMPLACER ( 255 = longueur maximum d'une chaîne)
: TROUVE?     ( str --- n)
    MOT-A-REMPLACER DOVER INSTR ;
```

n vaut 0 si MOT-A-REMPLACER ne figure pas dans (str), sinon sa valeur est la position de MOT-A-REMPLACER dans (str).

```
: REMPLACE-TOUT     ( str1 str2 str3 ---)
    MOT-A-REMPLACER $!     TROUVE?          ( Faut-il remplacer ??)
    IF BEGIN DOVER DOVER TROUVE?          ( TROUVE?  position)
             MOT-A-REMPLACER LEN MID$ SUB$!    ( remplacement)
             TROUVE? MOT                       ( Faut-il continuer ??)
       UNTIL
    THEN
    DDROP DDROP ;
```

Cette tournure est peu naturelle, il est plus correct d'utiliser WHILE (Voir Manuel de Référence : BEGIN.. WHILE.. REPEAT)

```
: REMPLACE-PARTOUT     ( str1 str2 str3 ---)
    MOT-A-REMPLACER $!
    BEGIN TROUVE?    ( Tant que MOT-A-REMPLACER figure dans str2)
    WHILE DOVER DOVER TROUVE?        ( En quelle position est-il ??)
          MOT-A-REMPLACER LEN MID$ SUB$!
    REPEAT
    DDROP DDROP ;
```



cedic nathan

INITIATION AU FORTH

Errata

Conventions :

1e colonne :	Numéro de page
2e colonne :	Numéro de la ligne. Les numéros négatifs indiquent une ligne à partir du bas. Ex : l.-5 Ligne 5 à partir du bas.

Dans certains exemples, le texte souligné est la réponse de l'ordinateur. Il indique dans le manuel un retour à la couleur noire lorsque ce qui précède était en bleu...

p.43	l.38	Manque « / » après « ELSE ». La ligne correcte est : ELSE /
p.44	l.7	Séparer « division avec mise en garde » de « n3 ». : /// (n1 n2 --- n3 division avec mise en garde)
p.44	l.-15	La parenthèse ouvrante est mal placée. Version correcte : ...et non lors de son exécution (Un tel mot s'appelle un mot *immédiat*. Cette question sera abordée à l'annexe C). Cela a pour principale conséquence...
p.103	l.-6	Manque un espace après « : », un autre après la parenthèse ouvrante. La ligne correcte est : : .U. (n ---) DUP .U. ; (Donne les deux affichages)
p.138	l.-14	Manque un espace entre LEFT$ et —TRAILING. La ligne correcte est : LIGNE$ 12 LEFT$ —TRAILING (Nom lu dans le répertoire)

p.147 l.-11	Le deuxième « A6B7 » (après le point-virgule) doit être en noir.
p.148 l.5	Séparer les crochets du mot HEX. La ligne correcte est : [HEX] (sortie provisoire du mode compilation)

Annexe A

p.124 l.-12	Remplacer « (en principe 4) » par « (4 ou 2 par défaut suivant que votre système a plus ou moins de 8 k en RAM) ».
p.125 l.6	Remplacer « sauf indication contraire... le contenu de 4 écrans » par « si on dispose de 4 buffers, on peut conserver en mémoire le contenu de 4 écrans ».
p.125 l.9	Remplacer « et aller jusqu'à 20 buffers en version cassette » par « et aller jusqu'à 22 buffers en version cassette, pour un système 24 k ».

L'impression de ce livre
a été réalisée sur les presses
des Imprimeries Aubin
à Poitiers/Ligugé

Achevé d'imprimer le 30 juillet 1984
N° d'impression, P 12897
Dépôt légal, juillet 1984

Imprimé en France